Лабораторная работа №3.1

Тема: Изучение метода пересчета мировых координат в оконные

Задание 1

В мировой системе координат (МСК) задана фигура. (рис.1)

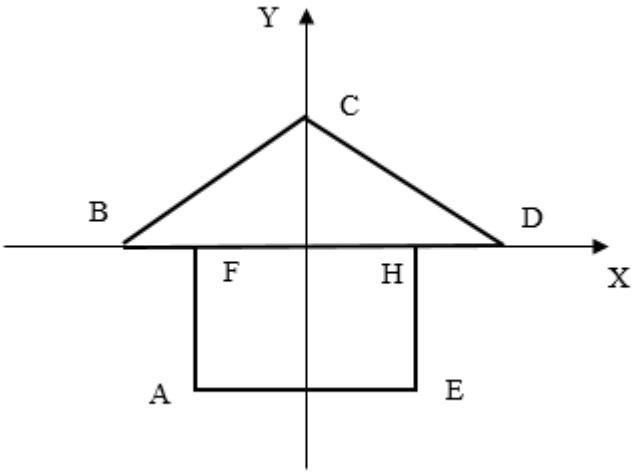


Рис.1

Координаты вершин:

	A	B	C	D	E	F	H
X	-2	-3	0	3	2	-2	2
Y	-4	0	4	0	-4	0	0

Отобразить фигуру в мировой системе координат.

Отобразить фигуру в прямоугольной области D^w окна Windows с координатами:

$$D^w = D^w\left(x_L^w, y_L^w, x_H^w, y_H^w\right) = D^w(100, 200, 400, 500),$$

где $\left(x_L^w, y_L^w\right)$ – координаты левого верхнего угла области D^w, $\left(x_H^w, y_H^w\right)$ – координаты правого нижнего угла области D^w.

Прямоугольную область в мировых координатах $D = D(x_L, y_H, x_H, y_L)$, где (x_L, y_H) – координаты левого верхнего угла области D, (x_H, y_L) – координаты правого нижнего угла области D, необходимую для

формирования матрицы пересчета координат из мировых в оконные, определить по габаритам фигуры ABCDE путем вычислений, т.е. положить $(x_L, y_H) = (x_{\min}, y_{\max})$, $(x_H, y_L) = (x_{\max}, y_{\min})$.

Задание 2

Построить график функции $f(x)$ для $x \subset [x_1; x_2]$ с шагом Δx в заданной прямоугольной области окна Windows $D^w = D^w\left(x_L^w, y_L^w, x_H^w, y_H^w\right)$, где $\left(x_L^w, y_L^w\right) = (100, 200)$ – координаты левого верхнего угла области D^w, $\left(x_H^w, y_H^w\right) = (800, 900)$ – координаты правого нижнего угла области D^w.
Прямоугольную область в мировых координатах $D = D(x_L, y_H, x_H, y_L)$, где (x_L, y_H) – координаты левого верхнего угла области D, (x_H, y_L) – координаты правого нижнего угла области D, необходимую для формирования матрицы пересчета координат из мировых в оконные, определить по габаритам графика, т.е. положить $(x_L, y_H) = (x_{\min}, y_{\max})$, $(x_H, y_L) = (x_{\max}, y_{\min})$

$$f(x) = \sin(\pi x)\sqrt{|x|}; \quad x_1 = -6; \; x_2 = 6; \; \Delta x = 0{,}1$$

Реализация задания №1

Записать данные таблицы в переменные:

$$Xa := -2 \quad Xb := -3 \quad Xc := 0 \quad Xd := 3 \quad Xe := 2 \quad Xf := -2 \quad Xg := 2$$
$$Ya := -4 \quad Yb := 0 \quad Yc := 4 \quad Yd := 0 \quad Ye := -4 \quad Yf := 0 \quad Yg := 0$$

Далее нужно записать левый верхний и правый нижний углы области в оконной системе координат.

Составляем вектора по данным нам значениям, находим левый верхний и правый нижний углы в мировой системе координат и вычисляем параметры, необходимые для формирования матрицы пересчета координат из МСК в ОСК:

$$Xlw := 100 \quad Ylw := 200$$
$$Xhw := 400 \quad Yhw := 500$$

$$Zx := (Xa, Xb, Xc, Xd, Xe, Xf, Xg)^T \quad Zy := (Ya, Yb, Yc, Yd, Ye, Yf, Yg)^T$$

$$dXw := Xhw - Xlw$$

$$Xl := \min(Zx) \quad Yh := \max(Zy) \quad dX := Xh - Xl$$

$$X := \max(Zx) \quad Yl := \min(Zy) \quad dYw := Yhw - Ylw$$

$$dY := Yh - Yl$$

$$Kx := \frac{dXw}{dX} \quad Ky := \frac{dYw}{dY}$$

Получаем все параметры для составления матрицы, которая выглядит следующим образом:

$$Tsw := \begin{matrix} Kx & 0 & Xlw - Kx * Xl \\ 0 & -Ky & Yhw + Ky * Yl \\ 0 & 0 & 1 \end{matrix}$$

Остается только:

1) вычислить координаты вершин в ОСК
2) построить вектора вершин в МСК и ОСК
3) построить графики в МСК и ОСК

1)Вычисление координат:

Всё, что нужно для этого сделать – мы сделали, т.е. нашли матрицу пересчета. Для того чтобы перевести координаты вершин в ОСК нужно каждый вектор вершин (A,B,C,D,E,F) умножить на нашу матрицу. Пример для вершины A:

$$\begin{pmatrix} Xaw \\ Yaw \\ q \end{pmatrix} := Tsw * \begin{pmatrix} Xa \\ Ya \\ 1 \end{pmatrix}$$

2)Вектора в МСК и ОСК:

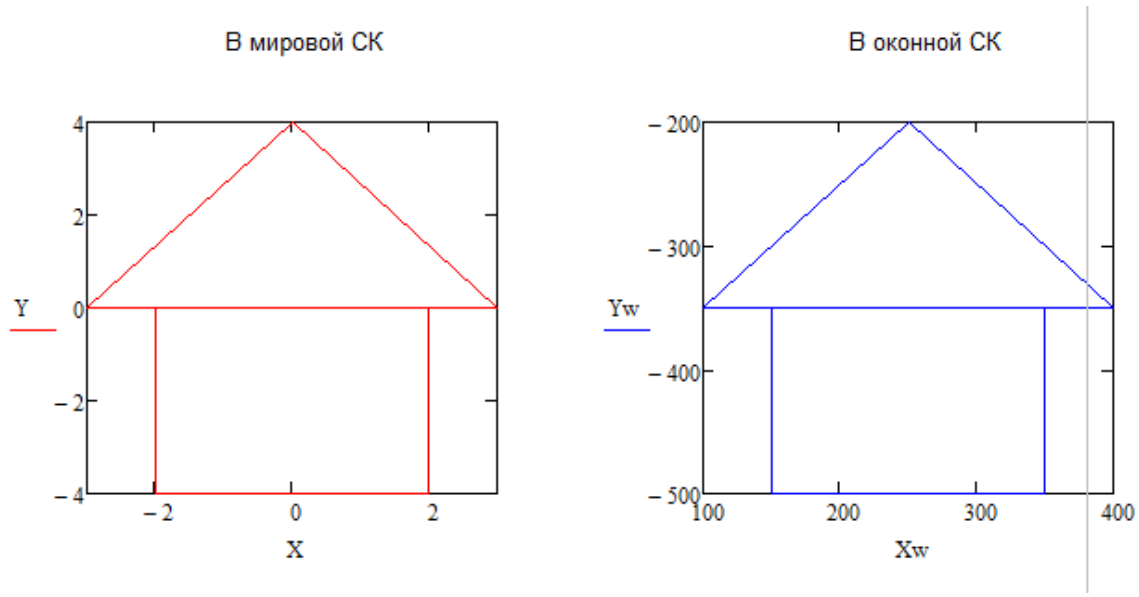
$$Xw := (Xbw, Xcw, Xdw, Xbw, Xfw, Xaw, Xew, Xgw, Xdw)^T$$

$$Yw := (Ybw, Ycw, Ydw, Ybw, Xfw, Xaw, Xew, Xgw, Xdw)^T$$

$$X := (Xb, Xc, Xd, Xb, Xf, Xa, Xe, Xg, Xd)^T$$

$$Y := (Yb, Yc, Yd, Yb, Yf, Ya, Ye, Yg, Yd)^T$$

3) И третье, последний шаг – построение графиков. По пройденному курсу MathCAD вы должны иметь представление о этой работе.



Реализация задания №2

Построить график функции f(x) с шагом dX для промежутка от x1 до x2

$f(x) := \sin(\pi \cdot x) \cdot \sqrt{|x|}$

$x1 := -6 \quad x2 := 6 \quad dX := 0.1$

Задаем область отображения графика в окне:

$Xlw := 100 \quad Ylw := 200$ - координаты левого верхнего угла

$Xhw := 800 \quad Yhw := 900$ - координаты правого нижнего угла

Вычисляем число точек графика

$N := \text{round}\left(\frac{x2 - x1}{dX}\right) = 120$

Заполняем массивы X,Y,Z значениями xi,yi=f(xi) и 1 соответственно в М С К

$i = 0..N$

$X_i := x1 + i \cdot dX$

$Y_i := f(X_i)$

$Z_i := 1$

- x, y - координаты точек графика в МСК

Определяем область графика в МСК

$Xl := \min(X) = -6$ $Yh := \max(Y) = 2.345$ -левый верхний угол в МСК

$Xh := \max(X) = 6$ $Yl := \min(Y) = -2.345$ - правый нижний угол в МСК

Формируем матрицу координат точек графика в МСК

$M := stack(X^T, Y^T, Z^T)$

Вычисляем параметры, необходимые для формирования матрицы пересчета координат из МСК в ОСК

$dXw := Xhw - Xlw = 700$ $dx := Xh - Xl = 12$

$dYw := Yhw - Ylw = 700$ $dy := Yh - Yl = 4.69$

$kx := \frac{dXw}{dx} = 58.333$ $ky := \frac{dYw}{dy} = 149.241$

Формируем матрицу пересчетов из МСК в ОСК

$$Tsw := \begin{pmatrix} kx & 0 & Xlw - kx * Xl \\ 0 & -ky & Yhw + ky * Yl \\ 0 & 0 & 1 \end{pmatrix}$$

Вычисляем матрицу координат точек графика в ОСК

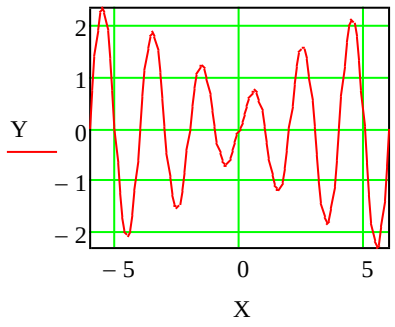
$Mw := Ysw * M$

Из матрицы M выбираем (x,y) - координаты точек графика для отображения

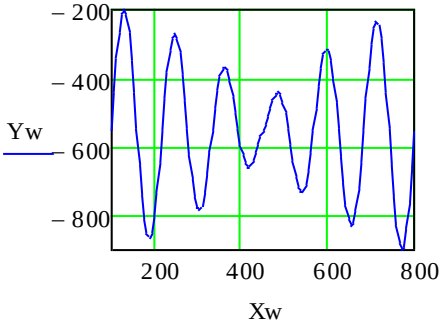
$Xw := \left(Mw^T\right)^{\langle 0 \rangle}$

$Yw := -\left(Mw^T\right)^{\langle 1 \rangle}$

В мировой СК



В оконной СК



Лабораторная работа № 3.2

Тема: Изучение режимов отображения и методов пересчета мировых координат в оконные

Задание:

1. Реализовать:

функцию
CMatrix SpaceToWindow(CRectD& rs,CRect& rw);
// Возвращает матрицу пересчета координат из мировых в оконные
// rs - область в мировых координатах - double
// rw - область в оконных координатах - int

функцию
void SetMyMode(CDC& dc,CRectD& RS,CRect& RW); //MFC
// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS - область в мировых координатах
// RW - Область в оконных координатах - int

структуру (для создания пера)
struct CMyPen

```
{
        int PenStyle;          // Стиль пера
        int PenWidth;          // Толщина пера
        COLORREF PenColor;     // Цвет пера
        CMyPen(){PenStyle=PS_SOLID; PenWidth=1;
PenColor=RGB(0,0,0);};
        void Set(int PS, int PW, COLORREF PC)
          {PenStyle=PS ; PenWidth=PW; PenColor=PC;};
};
```

класс (для отображения зависимости $Y_i=F(X_i)$

```
class CPlot2D
{
        CMatrix X;                      // Аргумент
        CMatrix Y;                      // Функция
        CMatrix K;                      // Матрица пересчета координат
        CRect RW;                       // Прямоугольник в окне
        CRectD RS;                      // Прямоугольник области в МСК
        CMyPen PenLine;                 // Перо для линий
        CMyPen PenAxis;                 // Перо для осей
 public:
        CPlot2D(){K.RedimMatrix(3,3);};        //Конструктор по умолчанию
```

```
        void SetParams(CMatrix& XX,CMatrix& YY,CRect& RWX); // Установка
                                                            // параметров графика
        void SetWindowRect(CRect& RWX);       //Установка области в окне для
отображения
                                                    //графика
        void GetWindowCoords(double xs,double ys, int &xw,int &yw); //Пересчет
                                                    //координаты точки из МСК в оконную
СК
        void SetPenLine(CMyPen& PLine); // Перо для рисования графика
        void SetPenAxis(CMyPen& PAxis);// Перо для осей координат
        void Draw(CDC& dc,int Ind1,int Int2);      // Рисование с самостоятельным
пересчетом
                                                    //координат
        void Draw1(CDC& dc,int Ind1,int Int2); // Рисование с БЕЗ
самостоятельного
                                                    //пересчета координат
        void GetRS(CRectD& RS); // Возвращает область графика в мировой СК
};
```

2. Создать приложение Windows ***MyPlot2D*** для графического отображения множества точек плоскости (x_i, y_i), $i = 0, 1, \ldots, N$, заданных в мировой системе координат (МСК), в прямоугольную области окна $D^w(x_L^w, y_L^w, x_H^w, y_H^w)$, где

   (x_L^w, y_L^w) – оконные координаты левого верхнего угла области D^w,

   (x_H^w, y_H^w) – оконные координаты правого нижнего угла области D^w.

   В классе ***CChildView*** приложения ***MyPlot2D*** реализовать функции

$$f_1(x) = \sin x / x \quad - \text{ double MyF1(double x)}$$
$$f_2(x) = x^3 \quad - \text{ double MyF2(double x)}$$
$$f_3(x) = \sqrt{x}\sin x \quad - \text{ double MyF3(double x)}$$
$$f_4(x) = x^2 \quad - \text{ double MyF4(double x)}$$

3. В приложении Windows ***MyPlot2D*** создать пункты меню:

   «Tests_F►F1»;
   «Tests_F►F2»;
   «Tests_F►F3»;
   «Tests_F►F4;
   «Tests_F►F1234».

4. Действия при выборе пункта меню «Tests_F►F1»:

Рассчитать значения функции $f_1(x)$ для $x \in [-3\pi; 3\pi]$ с шагом изменения аргумента $\Delta x = \pi/36$. В результате получить два массива – $Y_i = f_1(x_i)$ и X_i.

Установить параметры пера для отображения зависимости $Y_i = f_1(X_i)$ (толщина –***1***, цвет – ***красный***, тип линии – ***сплошная***)

Установить толщину пера для отображения координатных осей (толщина –***2***, цвет – ***синий***)

Установить параметры прямоугольной области для отображения графика в окне (координаты ***левого верхнего угла***, координаты ***правого нижнего угла***). Значения координат выбрать так, чтобы график располагался по центру окна.

Дать команду на отображение зависимости $Y_i = f_1(X_i)$ с установленными параметрами в режиме отображения **MM_TEXT .**

5. Действия при выборе пункта меню «Tests_F►F2»:

Рассчитать значения функции $f_2(x)$ для $x \in [-5; 5]$ с шагом изменения аргумента $\Delta x = 0{,}25$. В результате получить два массива – $Y_i = f_2(x_i)$ и X_i .

Установить параметры пера для отображения зависимости $Y_i = f_1(X_i)$ (толщина –***1***, цвет – ***зеленый***, тип линии – ***сплошная***)

Установить толщину пера для отображения координатных осей (толщина –***2***, цвет – ***синий***)

Установить параметры прямоугольной области для отображения графика в окне (координаты ***левого верхнего угла***, координаты ***правого нижнего угла***). Значения координат выбрать так, чтобы график располагался по центру окна.

Дать команду на отображение зависимости $Y_i = f_1(X_i)$ с установленными параметрами в режиме отображения **MM_ANISOTROPIC .**

6. Действия при выборе пункта меню «Tests_F►F3»:

Рассчитать значения функции $f_3(x)$ для $x \in [0; 6\pi]$ с шагом изменения аргумента $\Delta x = \pi/36$. В результате получить два массива – $Y_i = f_1(x_i)$ и X_i.

Установить параметры пера для отображения зависимости $Y_i = f_1(X_i)$ (толщина –***3***, цвет – ***красный***, тип линии – ***штрих - пунктирная***)

Установить толщину пера для отображения координатных осей (толщина –***2***, цвет – ***черный***)

Установить параметры прямоугольной области для отображения графика в окне (координаты ***левого верхнего угла***, координаты ***правого нижнего***

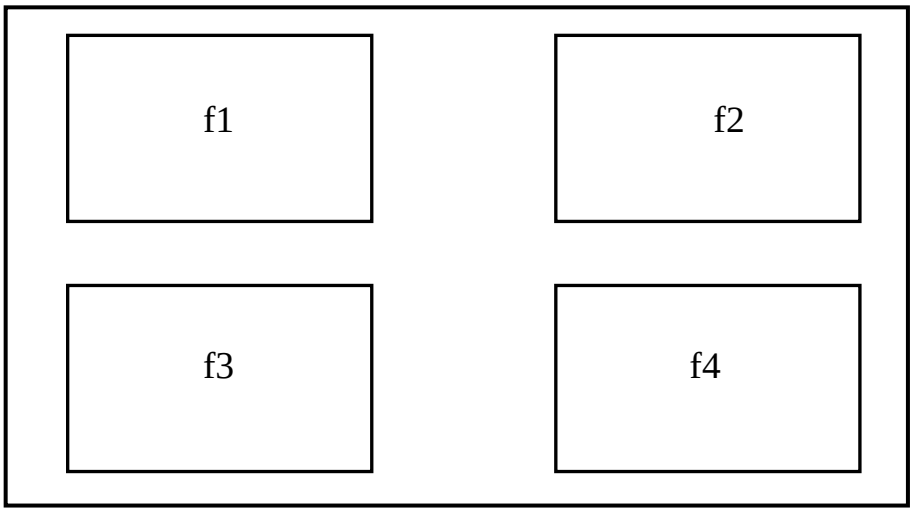
угла). Значения координат выбрать так, чтобы график располагался по центру окна.
Дать команду на отображение зависимости $Y_i = f_1(X_i)$ с установленными параметрами в режиме отображения **MM_TEXT** .

7. Действия при выборе пункта меню «Tests_F►F4»:

   Рассчитать значения функции $f_4(x)$ для $x \in [-10;10]$ с шагом изменения аргумента $\Delta x = 0{,}25$. В результате получить два массива – $Y_i = f_2(x_i)$ и X_i .
   Установить параметры пера для отображения зависимости $Y_i = f_1(X_i)$ (толщина –***2***, цвет – ***красный***, тип линии – ***сплошная***)
   Установить толщину пера для отображения координатных осей (толщина –***2***, цвет – ***синий***)
   Установить параметры прямоугольной области для отображения графика в окне (координаты ***левого верхнего угла***, координаты ***правого нижнего угла***). Значения координат выбрать так, чтобы график располагался по центру окна.
   Дать команду на отображение зависимости $Y_i = f_1(X_i)$ с установленными параметрами в режиме отображения **MM_ANISOTROPIC** .

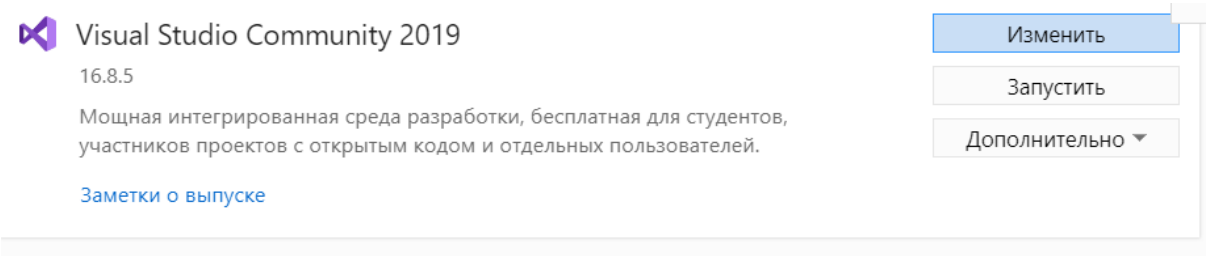
8. Действия при выборе пункта меню «Tests_F►F1234»:

   Дать команду на отображение графиков сразу четырех функций в режиме отображения **MM_TEXT**, которые к моменту выбора команды «Tests_F►F1234» должны быть созданы. Расположение как показано на рисунке. Размеры графиков должны быть одинаковыми.

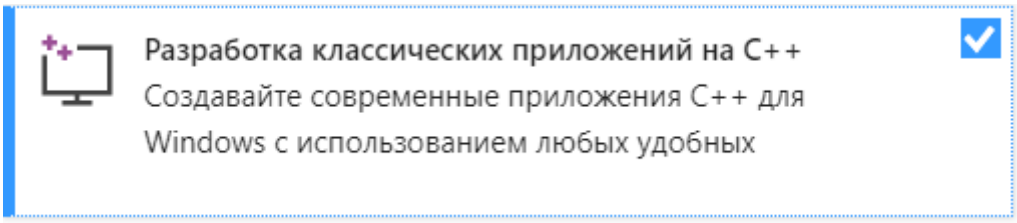


Для выполнения этой лабораторной работой, вам требуется установить библиотеку MFC.

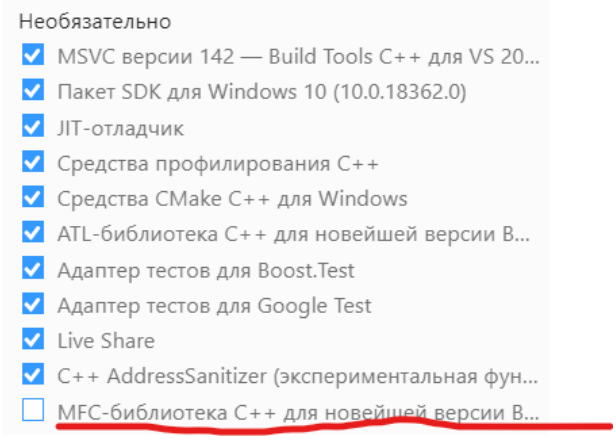
Заходим в Visual Studio Installer. Для добавления новых компонентов нажмите кнопку “Изменить”.
Заходим в Visual Studio Installer. Для добавления новых компонентов нажмите кнопку “Изменить”.



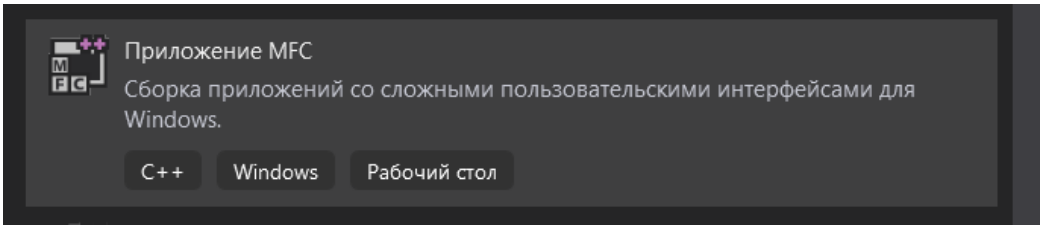
Выбираем пункт Разработка классических приложений на C++.



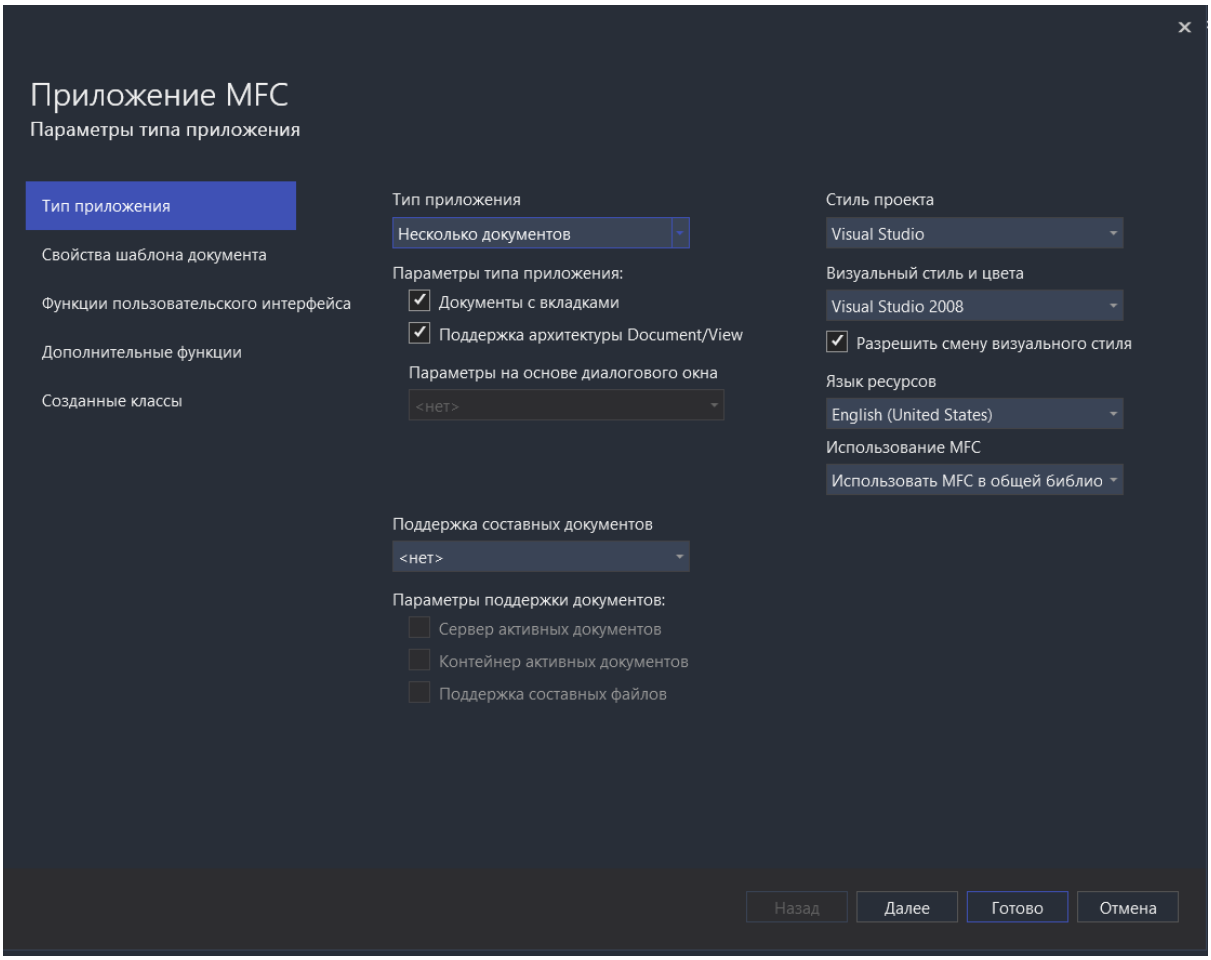
Далее в меню пакетов выбираем MFC-библиотеку C++.

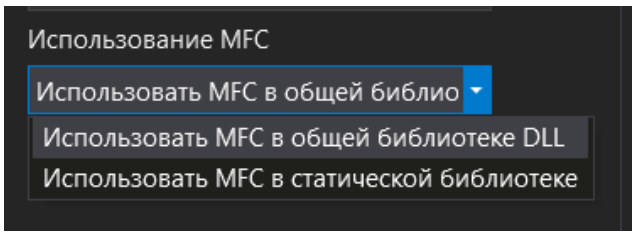


Создаём приложение MFC



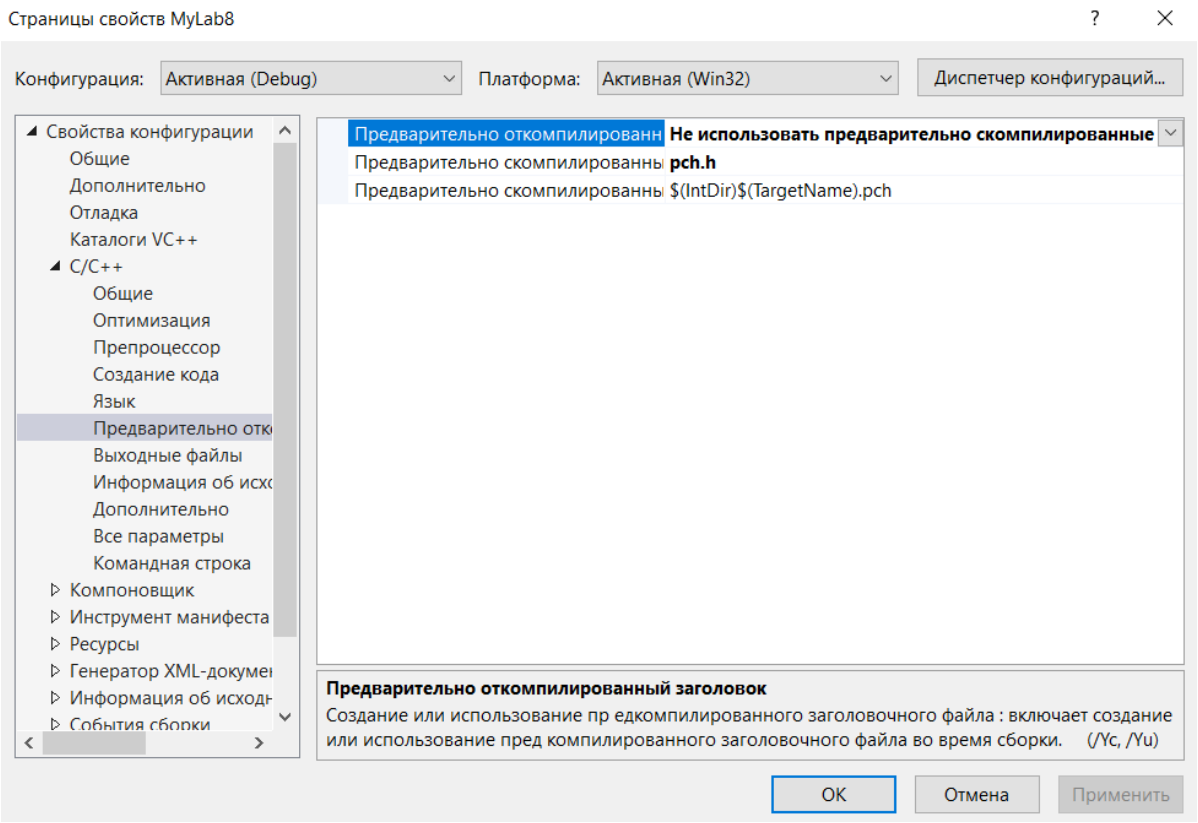
В настройках убеждаемся, что библиотека MFC будет использоваться в общей DLL



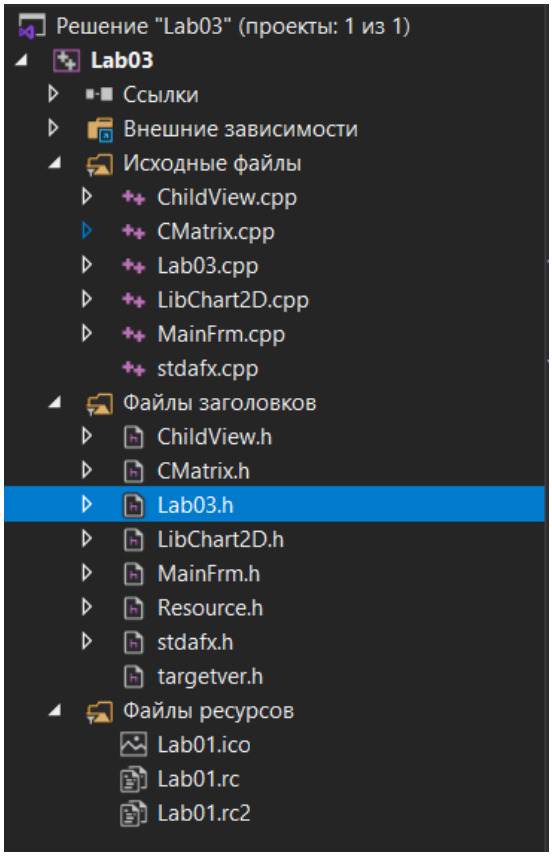


После удаления мусора, заходим в свойства проекта. Тыкаем на C/C++ и выбираем Предварительно. И ставим что мы НЕ БУДЕМ использовать этот pch.h. Мы будем использовать stdafx для мерджина.

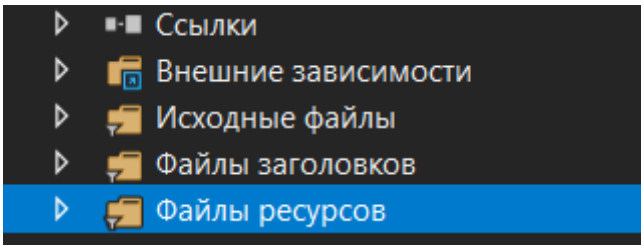
Теперь заходим в свойства проекта, C/C++, Предварительно откомпилированные и выбрать “Не использовать предварительно скомпилированные заголовки”.



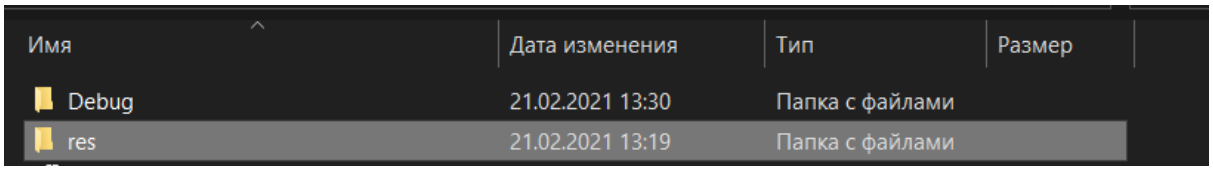
Чтобы начать работу над проектом, требуется удалить все созданные, при создании приложения, файлы.



Нам понадобятся файлы ресурсов.



Эти файлы располагаются в папке с проектом в папке res

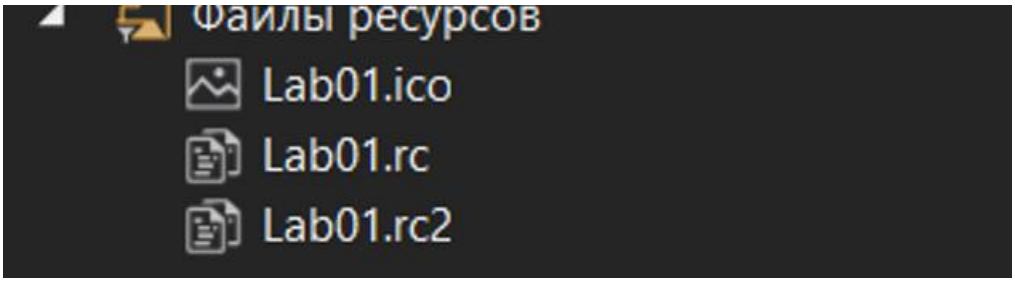


Если этой папки нет, то нужно ее создать. Или попробовать откомпилировать проект.

Изначально, после создания проекта, в ресурсах появится очень много файлов. Их все УДАЛЯЕМ, а не исключаем из проекта.

Вместе с этой лабораторной работой будет архив с нужными ресурсами. Их всего 3. Закидываем эти ресурсы в папку res, и добавляем в наш проект.

Получится вот так.



Их не надо менять, код, написанный далее. Он будет забинжен под эти ресурсы.

Далее создадим файл stdafx.h и stdafx.cpp, в котором будет служебная информация а также подключения нужных нам заголовочных файлов.

stdafx.h:

```
// stdafx.h: включите файл для добавления стандартных системных файлов
//или конкретных файлов проектов, часто используемых,
// но редко изменяемых
```

```
#pragma once

#ifndef VC_EXTRALEAN
#define VC_EXTRALEAN            // Исключите редко используемые
компоненты из заголовков Windows
#endif

#include "targetver.h"

#define _ATL_CSTRING_EXPLICIT_CONSTRUCTORS      // некоторые конструкторы
CString будут явными

// отключает функцию скрытия некоторых общих и часто пропускаемых
предупреждений MFC
#define _AFX_ALL_WARNINGS

#include <afxwin.h>         // основные и стандартные компоненты MFC
#include <afxext.h>         // расширения MFC

#include <FLOAT.H> // Для  DBL_MAX , DBL_MIN
#include <fstream>
#include <math.h>
#include "CMatrix.h"
#include <vector>

#include "LibGraph.h"



#ifndef _AFX_NO_OLE_SUPPORT
#include <afxdtctl.h>           // поддержка MFC для типовых элементов
управления Internet Explorer 4
#endif
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // поддержка MFC для типовых элементов
управления Windows
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxcontrolbars.h>     // поддержка MFC для лент и панелей
управления



#ifdef _UNICODE
#if defined _M_IX86
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='x86' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#elif defined _M_X64
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#else
#pragma comment(linker,"/manifestdependency:\"type='win32'
name='Microsoft.Windows.Common-Controls' version='6.0.0.0'
processorArchitecture='*' publicKeyToken='6595b64144ccf1df'
language='*'\"")
#endif
#endif
```

Большинство заголовочных файлов будет подчеркнуто красным. Реализацию этих файлов напишем позже.

Теперь напишем stdafx.cpp:

CMatrix.h:

```
#ifndef CMATRIXH
# define CMATRIXH 1
class CMatrix
{
    double** array;
    int n_rows;                          // Число строк
    int n_cols;                          // Число столбцов
public:
    CMatrix();                           // Конструктор по умолчанию (1 на 1)
    CMatrix(int, int);                   // Конструктор
    CMatrix(int);                        // Конструктор -вектора (один столбец)
    CMatrix(const CMatrix&);             // Конструктор копирования
    ~CMatrix();
    double& operator()(int, int);         // Выбор элемента матрицы по индексу
    double& operator()(int);             // Выбор элемента вектора по индексу
    CMatrix operator-();                 // Оператор "-"
    CMatrix operator=(const CMatrix&);   // Оператор "Присвоить":    M1=M2
    CMatrix operator*(CMatrix&);         // Оператор "Произведение": M1*M2
    CMatrix operator+(CMatrix&);         // Оператор "+": M1+M2
    CMatrix operator-(CMatrix&);         // Оператор "-": M1-M2
    CMatrix operator+(double);           // Оператор "+": M+a
    CMatrix operator-(double);           // Оператор "-": M-a
    CMatrix operator*(double);           // Оператор "-": M-a
    int rows()const { return n_rows; };   // Возвращает число строк
    int cols()const { return n_cols; };    // Возвращает число строк
    CMatrix Transp();                    // Возвращает матрицу,транспонированную к
текущей
    CMatrix GetRow(int);                 // Возвращает строку по номеру
    CMatrix GetRow(int, int, int);
    CMatrix GetCol(int);                 // Возвращает столбец по номеру
    CMatrix GetCol(int, int, int);
    CMatrix RedimMatrix(int, int);       // Изменяет размер матрицы с уничтожением
данных
    CMatrix RedimData(int, int);          // Изменяет размер матрицы с сохранением
данных,
                                         //которые можно сохранить
    CMatrix RedimMatrix(int);            // Изменяет размер матрицы с уничтожением
данных
    CMatrix RedimData(int);              // Изменяет размер матрицы с сохранением
данных,
                                         //которые можно сохранить
    double MaxElement();                 // Максимальный элемент матрицы
    double MinElement();                 // Минимальный элемент матрицы
};


#endif
```

CMatrix.cpp:

```
#include "stdafx.h"
//#include "CMatrix.h"

CMatrix::CMatrix()
{
    n_rows = 1;
    n_cols = 1;
    array = new double* [n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = 0;
}

//---------------------------------------------------------------------------
CMatrix::CMatrix(int Nrow, int Ncol)
// Nrow - число строк
// Ncol - число столбцов
{
    n_rows = Nrow;
    n_cols = Ncol;
    array = new double* [n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = 0;
}

//---------------------------------------------------------------------------
CMatrix::CMatrix(int Nrow)  //Вектор
// Nrow - число строк
{
    n_rows = Nrow;
    n_cols = 1;
    array = new double* [n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = 0;
}
//---------------------------------------------------------------------------
CMatrix::~CMatrix()
{
    for (int i = 0; i < n_rows; i++) delete array[i];
    delete array;
}

//---------------------------------------------------------------------------
double& CMatrix::operator()(int i, int j)
// i - номер строки
// j - номер столбца
{
    if ((i > n_rows - 1) || (j > n_cols - 1))     //  проверка выхода за диапазон
    {
        TCHAR* error = _T("CMatrix::operator(int,int): выход индекса за границу
диапазона ");
        MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    return array[i][j];
}

//---------------------------------------------------------------------------
```

```
double& CMatrix::operator()(int i)
// i - номер строки для вектора
{
    if (n_cols > 1)      //  Число столбцов больше одного
    {
        char* error = "CMatrix::operator(int): объект не вектор - число столбцов больше
1 ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    if (i > n_rows - 1)      //  проверка выхода за диапазон
    {
        TCHAR* error = TEXT("CMatrix::operator(int): выход индекса за границу диапазона
");
        MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    return array[i][0];
}
//---------------------------------------------------------------------------
CMatrix CMatrix::operator-()
// Оператор -M
{
    CMatrix Temp(n_rows, n_cols);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) = -array[i][j];
    return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator+(CMatrix& M)
// Оператор M1+M2
{
    int bb = (n_rows == M.rows()) && (n_cols == M.cols());
    if (!bb)
    {
        char* error = "CMatrix::operator(+): несоответствие размерностей матриц ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    CMatrix Temp(*this);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) += M(i, j);
    return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator-(CMatrix& M)
// Оператор M1-M2
{
    int bb = (n_rows == M.rows()) && (n_cols == M.cols());
    if (!bb)
    {
        char* error = "CMatrix::operator(-): несоответствие размерностей матриц ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    CMatrix Temp(*this);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) -= M(i, j);
    return Temp;
}
//---------------------------------------------------------------------------
```

```
CMatrix CMatrix::operator*(CMatrix& M)
// Умножение на матрицу M
{
    double sum;
    int nn = M.rows();
    int mm = M.cols();
    CMatrix Temp(n_rows, mm);
    if (n_cols == nn)
    {
        for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < mm; j++)
            {
                sum = 0;
                for (int k = 0; k < n_cols; k++) sum += (*this)(i, k) * M(k, j);
                Temp(i, j) = sum;
            }
    }
    else
    {
        TCHAR* error = TEXT("CMatrix::operator*: несоответствие размерностей матриц ");
        MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator=(const CMatrix& M)
// Оператор присваивания M1=M
{
    if (this == &M) return *this;
    int nn = M.rows();
    int mm = M.cols();
    if ((n_rows == nn) && (n_cols == mm))
    {
        for (int i = 0; i < n_rows; i++)
            for (int j = 0; j < n_cols; j++) array[i][j] = M.array[i][j];
    }
    else   // для ошибки размерностей
    {
        TCHAR* error = TEXT("CMatrix::operator=: несоответствие размерностей матриц");
        MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    return *this;
}

//---------------------------------------------------------------------------
CMatrix::CMatrix(const CMatrix& M) // Конструктор копирования
{
    n_rows = M.n_rows;
    n_cols = M.n_cols;
    array = new double* [n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = M.array[i][j];
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator+(double x)
// Оператор M+x, где M - матрица, x - число
{
    CMatrix Temp(*this);
```

```
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) += x;
    return Temp;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::operator*(double x)
// Оператор M*x, где M - матрица, x - число
{
    CMatrix Temp(*this);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) *= x;
    return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::operator-(double x)
// Оператор M+x, где M - матрица, x - число
{
    CMatrix Temp(*this);
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) Temp(i, j) -= x;
    return Temp;
}



//---------------------------------------------------------------------------
CMatrix CMatrix::Transp()
// Возвращает матрицу,транспонированную к (*this)
{
    CMatrix Temp(n_cols, n_rows);
    for (int i = 0; i < n_cols; i++)
        for (int j = 0; j < n_rows; j++) Temp(i, j) = array[j][i];
    return Temp;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::GetRow(int k)
// Возвращает строку матрицы по номеру k
{
    if (k > n_rows - 1)
    {
        char* error = "CMatrix::GetRow(int k): параметр k превышает число строк ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    CMatrix M(1, n_cols);
    for (int i = 0; i < n_cols; i++)M(0, i) = (*this)(k, i);
    return M;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::GetRow(int k, int n, int m)
// Возвращает подстроку из строки матрицы с номером k
// n - номер первого элемента в строке
// m - номер последнего элемента в строке
{
    int b1 = (k >= 0) && (k < n_rows);
    int b2 = (n >= 0) && (n <= m);
    int b3 = (m >= 0) && (m < n_cols);
    int b4 = b1 && b2 && b3;
    if (!b4)
    {
        char* error = "CMatrix::GetRow(int k,int n, int m):ошибка в параметрах ";
```

```
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    int nCols = m - n + 1;
    CMatrix M(1, nCols);
    for (int i = n; i <= m; i++)M(0, i - n) = (*this)(k, i);
    return M;
}

//---------------------------------------------------------------------------
CMatrix CMatrix::GetCol(int k)
// Возвращает столбец матрицы по номеру k
{
    if (k > n_cols - 1)
    {
        char* error = "CMatrix::GetCol(int k): параметр k превышает число столбцов ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    CMatrix M(n_rows, 1);
    for (int i = 0; i < n_rows; i++)M(i, 0) = (*this)(i, k);
    return M;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::GetCol(int k, int n, int m)
// Возвращает подстолбец из столбца матрицы с номером k
// n - номер первого элемента в столбце
// m - номер последнего элемента в столбце
{
    int b1 = (k >= 0) && (k < n_cols);
    int b2 = (n >= 0) && (n <= m);
    int b3 = (m >= 0) && (m < n_rows);
    int b4 = b1 && b2 && b3;
    if (!b4)
    {
        char* error = "CMatrix::GetCol(int k,int n, int m):ошибка в параметрах ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    int nRows = m - n + 1;
    CMatrix M(nRows, 1);
    for (int i = n; i <= m; i++)M(i - n, 0) = (*this)(i, k);
    return M;
}
//---------------------------------------------------------------------------
CMatrix CMatrix::RedimMatrix(int NewRow, int NewCol)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol - новое число столбцов
{
    for (int i = 0; i < n_rows; i++) delete array[i];
    delete array;
    n_rows = NewRow;
    n_cols = NewCol;
    array = new double* [n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = 0;
    return (*this);
}

//---------------------------------------------------------------------------
CMatrix CMatrix::RedimData(int NewRow, int NewCol)
```

```
// Изменяет размер матрицы с сохранением данных, которые можно сохранить
// NewRow - новое число строк
// NewCol - новое число столбцов
{
    CMatrix Temp = (*this);
    this->RedimMatrix(NewRow, NewCol);
    int min_rows = Temp.rows() < (*this).rows() ? Temp.rows() : (*this).rows();
    int min_cols = Temp.cols() < (*this).cols() ? Temp.cols() : (*this).cols();
    for (int i = 0; i < min_rows; i++)
        for (int j = 0; j < min_cols; j++) (*this)(i, j) = Temp(i, j);
    return (*this);
}

//---------------------------------------------------------------------------
CMatrix CMatrix::RedimMatrix(int NewRow)
// Изменяет размер матрицы с уничтожением данных
// NewRow - новое число строк
// NewCol=1
{
    for (int i = 0; i < n_rows; i++) delete array[i];
    delete array;
    n_rows = NewRow;
    n_cols = 1;
    array = new double* [n_rows];
    for (int i = 0; i < n_rows; i++) array[i] = new double[n_cols];
    for (int i = 0; i < n_rows; i++)
        for (int j = 0; j < n_cols; j++) array[i][j] = 0;
    return (*this);
}

//---------------------------------------------------------------------------
CMatrix CMatrix::RedimData(int NewRow)
// Изменяет размер матрицы с сохранением данных, которые можно сохранить
// NewRow - новое число строк
// NewCol=1
{
    CMatrix Temp = (*this);
    this->RedimMatrix(NewRow);
    int min_rows = Temp.rows() < (*this).rows() ? Temp.rows() : (*this).rows();
    for (int i = 0; i < min_rows; i++)(*this)(i) = Temp(i);
    return (*this);
}
//---------------------------------------------------------------------------
double CMatrix::MaxElement()
// Максимальное значение элементов матрицы
{
    double max = (*this)(0, 0);
    for (int i = 0; i < (this->rows()); i++)
        for (int j = 0; j < (this->cols()); j++)
            if ((*this)(i, j) > max) max = (*this)(i, j);
    return max;
}

//---------------------------------------------------------------------------
double CMatrix::MinElement()
// Минимальное значение элементов матрицы
{
    double min = (*this)(0, 0);
    for (int i = 0; i < (this->rows()); i++)
        for (int j = 0; j < (this->cols()); j++)
            if ((*this)(i, j) < min) min = (*this)(i, j);
    return min;
}
```

Добавим MainFrm, это компонент класса CMainFrame
MainFrm.h:

```
// MainFrm.h: интерфейс класса CMainFrame
//

#pragma once
#include "ChildView.h"

class CMainFrame : public CFrameWnd
{

public:
    CMainFrame() noexcept;
protected:
    DECLARE_DYNAMIC(CMainFrame)

    // Атрибуты
public:

    // Операции
public:

    // Переопределение
public:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo);

    // Реализация
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // встроенные члены панели элементов управления
    CStatusBar        m_wndStatusBar;
    CChildView    m_wndView;

    // Созданные функции схемы сообщений
protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    DECLARE_MESSAGE_MAP()

};
```

MainFrm.cpp:

```
// MainFrm.cpp: реализация класса CMainFrame
//

#include "stdafx.h"
```

```
#include "MyLab8.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    ON_WM_CREATE()
    ON_WM_SETFOCUS()
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // индикатор строки состояния
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

// Создание или уничтожение CMainFrame

CMainFrame::CMainFrame() noexcept
{
    // TODO: добавьте код инициализации члена
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    //--------------------------------------------------------------------
    int width = 450, height = 500;
    MoveWindow((GetSystemMetrics(SM_CXSCREEN) / 2 - width / 2),
        (GetSystemMetrics(SM_CYSCREEN) / 2 - height / 2), width, height);
    //--------------------------------------------------------------------


    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    // создать представление для размещения рабочей области рамки
    if (!m_wndView.Create(nullptr, nullptr, AFX_WS_DEFAULT_VIEW, CRect(0, 0, 0, 0), this,
AFX_IDW_PANE_FIRST, nullptr))
    {
        TRACE0("Не удалось создать окно представлений\n");
        return -1;
    }

    if (!m_wndStatusBar.Create(this))
    {
        TRACE0("Не удалось создать строку состояния\n");
        return -1;      // не удалось создать
    }
    m_wndStatusBar.SetIndicators(indicators, sizeof(indicators) / sizeof(UINT));
```

```
    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CFrameWnd::PreCreateWindow(cs))
        return FALSE;
    // TODO: изменить класс Window или стили посредством изменения
    //  CREATESTRUCT cs

    cs.style = WS_OVERLAPPED | WS_CAPTION | FWS_ADDTOTITLE
        | WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX;

    cs.dwExStyle &= ~WS_EX_CLIENTEDGE;
    cs.lpszClass = AfxRegisterWndClass(0);
    return TRUE;
}

// Диагностика CMainFrame

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}
#endif //_DEBUG


// Обработчики сообщений CMainFrame

void CMainFrame::OnSetFocus(CWnd* /*pOldWnd*/)
{
    // передача фокуса окну представления
    m_wndView.SetFocus();
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void* pExtra, AFX_CMDHANDLERINFO*
pHandlerInfo)
{
    // разрешить ошибки в представлении при выполнении команды
    if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
        return TRUE;

    // в противном случае выполняется обработка по умолчанию
    return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}
```

ChildView.h:

```
// ChildView.h: интерфейс класса CChildView
//
```

```
#pragma once


// Окно CChildView

class CChildView : public CWnd
{
    // Создание
public:
    CChildView();

    // Атрибуты
public:
    CRect WinRect;
    CMatrix PView;
    CMatrix PLight;
    int Index;

    // Операции
public:

    // Переопределение
protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

    // Реализация
public:
    virtual ~CChildView();

    // Созданные функции схемы сообщений
protected:
    afx_msg void OnPaint();
    DECLARE_MESSAGE_MAP()
public:
    // действия при выборе пункта меню
    afx_msg void OnSphere_Mirror();
    afx_msg void OnSphere_Diffuse();
    afx_msg void OnSize(UINT nType, int cx, int cy);
};
```

ChildView.cpp:

```
// ChildView.cpp: реализация класса CChildView
//

#include "stdafx.h"
#include "MyLab8.h"
#include "ChildView.h"
#include "LibLabs3D.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CChildView
//COLORREF Color = RGB(255, 0, 0);

CChildView::CChildView()
```

```
{
    Index = 3;
    PView.RedimMatrix(3);
    PLight.RedimMatrix(3);
}

CChildView::~CChildView()
{
}

// Реализация карты сообщений
BEGIN_MESSAGE_MAP(CChildView, CWnd)
    ON_WM_PAINT()
    // сообщения меню выбора

    ON_COMMAND(ID_Sphere_Mirror, &CChildView::OnSphere_Mirror)
    ON_COMMAND(ID_Sphere_Diffuse, &CChildView::OnSphere_Diffuse)
    ON_WM_SIZE()
END_MESSAGE_MAP()



// Обработчики сообщений CChildView

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
    if (!CWnd::PreCreateWindow(cs))
        return FALSE;

    cs.dwExStyle |= WS_EX_CLIENTEDGE;
    cs.style &= ~WS_BORDER;
    cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS,
        ::LoadCursor(nullptr, IDC_ARROW), reinterpret_cast<HBRUSH>(COLOR_WINDOW + 1),
nullptr);

    return TRUE;
}

void CChildView::OnPaint()
{
    CPaintDC dc(this); // контекст устройства для рисования

    if (Index == 0)
        DrawLightSphere(dc, 1, PView, PLight, WinRect, RGB(255, 0, 0), Index);
    if (Index == 1)
        DrawLightSphere(dc, 1, PView, PLight, WinRect, RGB(255, 0, 0), Index);

}

void CChildView::OnSphere_Mirror()
{
    Index = 1;
    Invalidate();
    PView(0) = 100; PView(1) = 0;  PView(2) = 60;
}

void CChildView::OnSphere_Diffuse()
{
    Index = 0;
    Invalidate();
    PView(0) = 100; PView(1) = 0;  PView(2) = 60;
}
```

```
void CChildView::OnSize(UINT nType, int cx, int cy)
{
    CWnd::OnSize(nType, cx, cy);                // для динамического изменения окна
    WinRect.SetRect(50, 50, cx - 50, cy - 50);  // параметры окна рисования
}
```

Задание:

1. Реализовать:

функцию

CMatrix SpaceToWindow(CRectD& rs,CRect& rw);

// Возвращает матрицу пересчета координат из мировых в оконные
// rs - область в мировых координатах - double
// rw - область в оконных координатах - int

функцию

void SetMyMode(CDC& dc,CRectD& RS,CRect& RW); //MFC

// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS - область в мировых координатах
// RW - Область в оконных координатах - int

структуру (для создания пера)

```
struct CMyPen
{
int PenStyle;                   // Стиль пера
int PenWidth;                  // Толщина пера
COLORREF PenColor;    // Цвет пера
         CMyPen(){PenStyle=PS_SOLID; PenWidth=1;
PenColor=RGB(0,0,0);};
         void Set(int PS, int PW, COLORREF PC)
                  {PenStyle=PS ; PenWidth=PW; PenColor=PC;};
};
```

класс (для отображения зависимости $Y_i=F(X_i)$:

```
class CPlot2D
{
CMatrix X;                                  // Аргумент
CMatrix Y;                                  // Функция
CMatrix K;                                  // Матрица пересчета координат
CRect RW;                                   // Прямоугольник в окне
CRectD RS;                                  // Прямоугольник области в МСК
CMyPen PenLine;                             // Перо для линий
CMyPen PenAxis;                             // Перо для осей
 public:
CPlot2D(){K.RedimMatrix(3,3);};          //Конструктор по умолчанию
void SetParams(CMatrix& XX,CMatrix& YY,CRect& RWX); // Установка
```

```
// параметров графика
void SetWindowRect(CRect& RWX); //Установка области в окне для
отображения графика
void GetWindowCoords(double xs,double ys, int &xw,int &yw);
//Пересчет координаты точки из МСК в оконную СК
void SetPenLine(CMyPen& PLine); // Перо для рисования графика
void SetPenAxis(CMyPen& PAxis);       // Перо для осей координат
void Draw(CDC& dc,int Ind1,int Int2); // Рисование с самостоятельным
пересчетом                                    //координат
void Draw1(CDC& dc,int Ind1,int Int2); // Рисование с БЕЗ
самостоятельного пересчета координат
void GetRS(CRectD& RS);          // Возвращает область графика в
мировой СК
};
```

2. Создать приложение Windows ***MyPlot2D*** для графического отображения множества точек плоскости (x_i, y_i), $i = 0, 1, \ldots, N$, заданных в мировой системе координат (МСК), в прямоугольную области окна $D^w(x_L^w, y_L^w, x_H^w, y_H^w)$, где

(x_L^w, y_L^w) – оконные координаты левого верхнего угла области D^w,
(x_H^w, y_H^w) – оконные координаты правого нижнего угла области D^w.
В классе ***CChildView*** приложения ***MyPlot2D*** реализовать функции

$f_1(x) = \sin x / x$ – double MyF1(double x)
$f_2(x) = x^3$ – double MyF2(double x)
$f_3(x) = \sqrt{x}\sin x$ – double MyF3(double x)
$f_4(x) = x^2$ – double MyF4(double x)

3. В приложении Windows ***MyPlot2D*** создать пункты меню:

«Tests_F►F1»;
«Tests_F►F2»;
«Tests_F►F3»;
«Tests_F►F4;
«Tests_F►F1234».

4. Действия при выборе пункта меню «Tests_F►F1»:

Рассчитать значения функции $f_1(x)$ для $x \in [-3\pi; 3\pi]$ с шагом изменения аргумента $\Delta x = \pi/36$. В результате получить два массива – $Y_i = f_1(x_i)$ и X_i.

Установить параметры пера для отображения зависимости $Y_i = f_1(X_i)$ (толщина –***1***, цвет – ***красный***, тип линии – ***сплошная***)
Установить толщину пера для отображения координатных осей (толщина –***2***, цвет – ***синий***)
Установить параметры прямоугольной области для отображения графика в окне (координаты ***левого верхнего угла***, координаты ***правого нижнего угла***). Значения координат выбрать так, чтобы график располагался по центру окна.
Дать команду на отображение зависимости $Y_i = f_1(X_i)$ с установленными параметрами в режиме отображения **MM_TEXT** .

5. Действия при выборе пункта меню «Tests_F►F2»:

Рассчитать значения функции $f_2(x)$ для $x \in [-5; 5]$ с шагом изменения аргумента $\Delta x = 0{,}25$. В результате получить два массива – $Y_i = f_2(x_i)$ и X_i.
Установить параметры пера для отображения зависимости $Y_i = f_1(X_i)$ (толщина –***1***, цвет – ***зеленый***, тип линии – ***сплошная***)
Установить толщину пера для отображения координатных осей (толщина –***2***, цвет – ***синий***)
Установить параметры прямоугольной области для отображения графика в окне (координаты ***левого верхнего угла***, координаты ***правого нижнего угла***). Значения координат выбрать так, чтобы график располагался по центру окна.
Дать команду на отображение зависимости $Y_i = f_1(X_i)$ с установленными параметрами в режиме отображения **MM_ANISOTROPIC** .

6. Действия при выборе пункта меню «Tests_F►F3»:

Рассчитать значения функции $f_3(x)$ для $x \in [0; 6\pi]$ с шагом изменения аргумента $\Delta x = \pi/36$. В результате получить два массива – $Y_i = f_1(x_i)$ и X_i.
Установить параметры пера для отображения зависимости $Y_i = f_1(X_i)$ (толщина –***3***, цвет – ***красный***, тип линии – ***штрих - пунктирная***)
Установить толщину пера для отображения координатных осей (толщина –***2***, цвет – ***черный***)
Установить параметры прямоугольной области для отображения графика в окне (координаты ***левого верхнего угла***, координаты ***правого нижнего угла***). Значения координат выбрать так, чтобы график располагался по центру окна.
Дать команду на отображение зависимости $Y_i = f_1(X_i)$ с установленными параметрами в режиме отображения **MM_TEXT** .

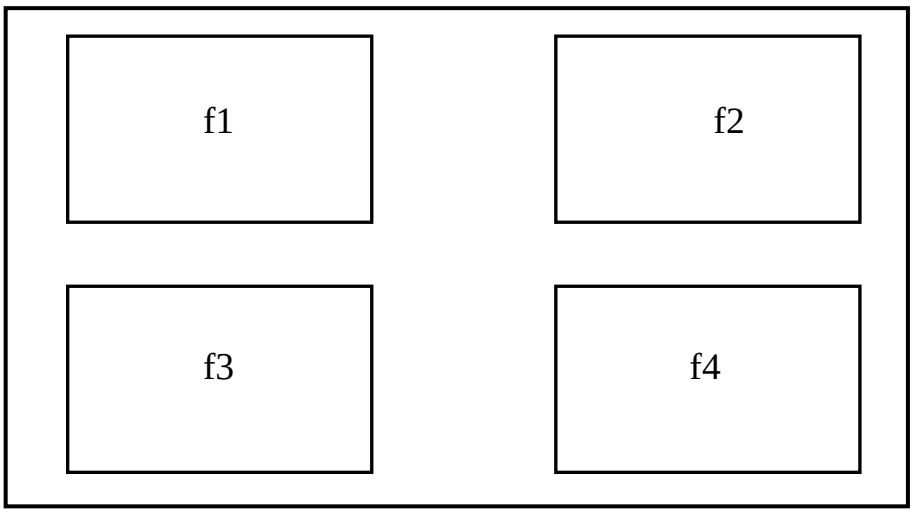
7. Действия при выборе пункта меню «Tests_F►F4»:

Рассчитать значения функции $f_4(x)$ для $x \in [-10; 10]$ с шагом изменения аргумента $\Delta x = 0{,}25$. В результате получить два массива – $Y_i = f_2(x_i)$ и X_i.

Установить параметры пера для отображения зависимости $Y_i = f_1(X_i)$ (толщина –***2***, цвет – ***красный***, тип линии – ***сплошная***)
Установить толщину пера для отображения координатных осей (толщина –***2***, цвет – ***синий***)
Установить параметры прямоугольной области для отображения графика в окне (координаты ***левого верхнего угла***, координаты ***правого нижнего угла***). Значения координат выбрать так, чтобы график располагался по центру окна.
Дать команду на отображение зависимости $Y_i = f_1(X_i)$ с установленными параметрами в режиме отображения **MM_ANISOTROPIC** .

8. Действия при выборе пункта меню «Tests_F►F1234»:

Дать команду на отображение графиков сразу четырех функций в режиме отображения **MM_TEXT**, которые к моменту выбора команды «Tests_F►F1234» должны быть созданы. Расположение как показано на рисунке. Размеры графиков должны быть одинаковыми.



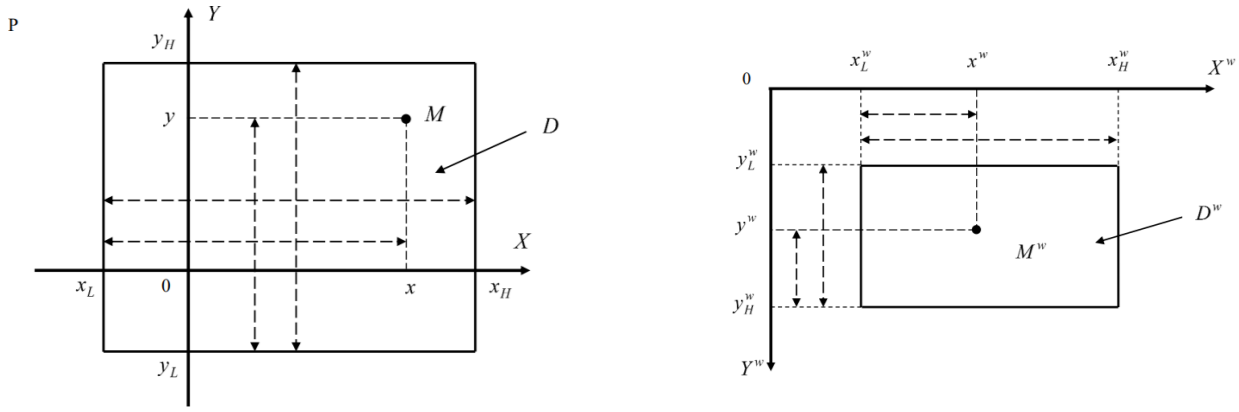
Ход работы:

1. Для выполнения первого задания данной лабораторной работы необходимо:

CMatrix SpaceToWindow:

Рассмотрим оконную систему координат X^wOY^w с началом в левом верхнем углу окна и выделим в ней некоторую прямоугольную область D^w

M = M(x, y) – некоторая точка в мировой системе координат, а $M^w = M^w(x^w, y^w)$ ее образ в оконной системе координат.



Из приведенных рисунков можно получить пропорции:

$$\begin{cases} \dfrac{x - x_L}{x_H - x_L} = \dfrac{x^w - x_L^w}{x_H^w - x_L^w} \\ \dfrac{y - y_L}{y_H - y_L} = -\dfrac{y^w - y_L^w}{y_H^w - y_L^w} \end{cases}$$

$x_H^w - x_L^w = \Delta x^w$ – ширина области отображения в оконных координатах,

$x_H - x_L = \Delta x$ – ширина области отображения в мировых координатах,

$y_H^w - y_L^w = \Delta y^w$ – высота области отображения в оконных координатах,

$y_H - y_L = \Delta y$ – высота области отображения в мировых координатах.

Выделим x^w и y^w:

$$\begin{cases} x^w = x_L^w + k_x(x - x_L) \\ y^w = y_H^w - k_y(y - y_L) \end{cases},$$

Приведем систему к матричному виду:

$$\begin{pmatrix} x^w \\ y^w \\ 1 \end{pmatrix} = \begin{pmatrix} k_x & 0 & x_L^w \\ 0 & -k_y & y_H^w \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x - x_L \\ y - y_L \\ 1 \end{pmatrix}$$

где

$$k_x = \frac{\Delta x^w}{\Delta x} \quad k_y = \frac{\Delta y^w}{\Delta y}$$

или

$$V^w = T_1 V_1$$

где T_1 и есть искомая матрица

Код:

```
CMatrix SpaceToWindow(CRectD& RS, CRect& RW)
// Возвращает матрицу пересчета координат из мировых в оконные
// RS - область в мировых координатах - double
// RW - область в оконных координатах - int
{
        CMatrix M(3, 3);
        CSize sz = RW.Size();    // Размер области в ОКНЕ
        int dwx = sz.cx;        // Ширина
        int dwy = sz.cy;        // Высота
        CSizeD szd = RS.SizeD(); // Размер области в МИРОВЫХ
координатах

        double dsx = szd.cx;    // Ширина в мировых координатах
        double dsy = szd.cy;    // Высота в мировых координатах

        double kx = (double)dwx / dsx;   // Масштаб по x
        double ky = (double)dwy / dsy;   // Масштаб по y

        M(0, 0) = kx;  M(0, 1) = 0;    M(0, 2) = (double)RW.left - kx
* RS.left;
        M(1, 0) = 0;   M(1, 1) = -ky;  M(1, 2) = (double)RW.bottom +
ky * RS.bottom;
        M(2, 0) = 0;   M(2, 1) = 0;    M(2, 2) = 1;
        return M;
}
```

void SetMyMode(CDC& dc,CRectD& RS,CRect& RW) :

Режим **MM_ANISOTROPIC** (анизотропный) допускает изменение направления осей X и Y, а также изменение масштаба осей координат. Анизотропный режим **MM_ANISOTROPIC** предполагает использование разных масштабов для разных осей (хотя можно использовать и одинаковые масштабы).

Для изменения ориентации и масштаба осей можно воспользоваться функциями **SetViewportExt** и **SetWindowExt**, которые являются методами класса CDC из библиотеки MFC:

```
virtual CSize SetWindowExt(int cx, int cy),
virtual CSize SetViewportExt(int cx, int cy).
```

Функция **SetWindowExt** устанавливает для формулы преобразования координат значения переменных **xWinExt** и **yWinExt** . Функция **SetViewportExt** устанавливает для формулы преобразования координат значения переменных **xViewExt** и **yViewExt** .

Функция **SetViewportExt** должна использоваться после функции **SetWindowExt**.

Анизотропный режим удобен в тех случаях, когда изображение должно занимать всю внутреннюю поверхность окна при любом изменении размеров окна. Соотношение масштабов при этом не сохраняется.

$$\begin{cases} xViewport = (xWindow - xWinOrg)\dfrac{xViewExt}{xWinExt} + xViewOrg \\ yViewport = (yWindow - yWinOrg)\dfrac{yViewExt}{yWinExt} + yViewOrg \end{cases}$$

Формулы, по которым пересчитывают координаты функции GDIWindows:

Возьмем выражения для пересчета координат из мировых в оконные:

$$\begin{cases} x^w = x_L^w + \dfrac{\Delta x^w}{\Delta x}(x - x_L) \\ y^w = y_H^w - \dfrac{\Delta y^w}{\Delta y}(y - y_L) \end{cases}$$

Отсюда получаем:

$$yWinExt = \Delta y, \quad yViewExt = -\Delta y^w,$$
$$xWinOrg = x_L, \quad xViewOrg = x_L^w,$$
$$yWinOrg = y_L, \quad yViewOrg = y_H^w,$$

$$xWinExt = \Delta x, \quad xViewExt = \Delta x^w$$

Что означает: перевод координат из мировых в оконные можно возложить на функции **GDI Windows**, установив предварительно режим отображения **MM_ANISOTROPIC**

Код:

```
void SetMyMode(CDC& dc, CRectD& RS, CRect& RW)  //MFC
// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS -  область в мировых координатах - int
// RW -   Область в оконных координатах - int
{
        double dsx = RS.right - RS.left;
        double dsy = RS.top - RS.bottom;
        double xsL = RS.left;
        double ysL = RS.bottom;

        int dwx = RW.right - RW.left;
        int dwy = RW.bottom - RW.top;
        int xwL = RW.left;
        int ywH = RW.bottom;

        dc.SetMapMode(MM_ANISOTROPIC);     // задание режима
отображения координат
        dc.SetWindowExt(dsx, dsy);         // размер окна вдоль двух
логических координат
        dc.SetViewportExt(dwx, -dwy);      // размер окна вдоль двух
физических координат
        dc.SetWindowOrg(xsL, ysL);         // задание начала
отображения в логической системе координат
        dc.SetViewportOrg(xwL, ywH); // задание начала отображения в
физической системе координат
}


struct CMyPen
{
     int PenStyle;                    // Стиль пера
     int PenWidth;                    // Толщина пера
     COLORREF PenColor;               // Цвет пера
     CMyPen() { PenStyle = PS_SOLID; PenWidth = 1; PenColor = RGB(0,
0, 0); };
     void Set(int PS, int PW, COLORREF PC) { PenStyle = PS; PenWidth =
PW; PenColor = PC; };
};
```

Это – структура **CMyPen**, которая и представляет собой перо, которым мы будем рисовать наши графики. Она содержит три поля, которые отвечают за стиль, толщину и цвет, конструктор по умолчанию и метод **Set()**, благодаря

которому мы можем сами установить стиль нашего пера. Мы будем работать с двумя экземплярами этой структуры: **PenLine** – перо, для рисования графика и **PenAxis** – перо, для рисования осей.

Таким образом, в нашей функции **OnTest1**() строка **PenLine.Set**(PS_SOLID, 1, RGB(255, 0, 0)) говорит о том, что наше перо будет рисовать сплошной линией красного цвета толщиной 1 и **PenAxis.Set**(PS_SOLID, 2, RGB(0, 0, 255)) – перо, которое рисует сплошной синей линией толщиной 2.

```
class CPlot2D
{
	CMatrix X;			// Аргумент
	CMatrix Y;			// Функция
	CMatrix K;			// Матрица пересчета координат
	CRect RW;			// Прямоугольник в окне
	CRectD RS;			// Прямоугольник области в МСК
	CMyPen PenLine;		// Перо для линий
	CMyPen PenAxis;		// Перо для осей
public:
	CPlot2D() { K.RedimMatrix(3, 3); };		//Конструктор по умолчанию
	void SetParams(CMatrix& XX, CMatrix& YY, CRect& RWX); // Установка параметров графика
	void SetWindowRect(CRect& RWX);			//Установка области в окне для отображения графика
	void GetWindowCoords(double xs, double ys, int &xw, int &yw);
	//Пересчет координаты точки из МСК в оконную СК
	void SetPenLine(CMyPen& PLine);			// Перо для рисования графика
	void SetPenAxis(CMyPen& PAxis);			// Перо для осей координат
	void Draw(CDC& dc, int Ind1, int Int2);		// Рисование с самостоятельным пересчетом координат
	void Draw1(CDC& dc, int Ind1, int Int2);	// Рисование БЕЗ самостоятельного пересчета координат
	void DrawBezier(CDC& dc, int NT);
	void GetRS(CRectD& RS);// Возвращает область графика в мировой СК
};
```

Метод **SetParams** устанавливает параметры графика и принимает 3 параметра:

CMatrix& XX - вектор данных по X
CMatrix& YY - вектор данных по Y
CRect& RWX - область в окне

Код:

```
void CPlot2D::SetParams(CMatrix& XX, CMatrix& YY, CRect& RWX)
{
	int nRowsX = XX.rows();
```

```
	int nRowsY = YY.rows();
	if (nRowsX != nRowsY)
	{
		TCHAR* error = _T("SetParams: неправильные размеры массивов данных");
		MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);

		exit(1);
	}
	X.RedimMatrix(nRowsX); // Изменяет размер матрицы с уничтожением данных
//параметр – новое число строк
	Y.RedimMatrix(nRowsY); // аналогично
	X = XX;
	Y = YY;
//Максимальные и минимальные значения матриц
	double x_max = X.MaxElement();
	double x_min = X.MinElement();
	double y_max = Y.MaxElement();
	double y_min = Y.MinElement();
	RS.SetRectD(x_min, y_max, x_max, y_min);// Область в мировой СК
	RW.SetRect(RWX.left, RWX.top, RWX.right, RWX.bottom); // Область в окне
	K = SpaceToWindow(RS, RW); // Матрица пересчета координат
	// SpaceToWindow - Возвращает матрицу пересчета координат из мировых в оконные
// RS - область в мировых координатах - double
// RW - область в оконных координатах - int
}
```

Метод **SetWindowRect** устанавливает область в окне отображения графика.
Метод **GetWindowCoords** пересчитывает координаты из МСК в оконную СК
Метод **SetPenLine** и **SetPenAxis** установка параметров пера для линии графика и установка параметров пера для линий осей сотвественно
Метод **Draw** рисует график в режиме MM_TEXT - собственный пересчет координат
Метод **Draw1** рисует график в режиме MM_ANISOTROPIC - без собственного пересчета координат
Метод **DrawBezier** рисует кривую Безье по набору опорных точек
Метод **GetRS** - RS - структура, куда записываются параметры области графика

2. Для выполнения второго задания данной лабораторной работы необходимо:

Мировые координаты объектов являются трехмерными. Положение объекта может быть описано, например, в прямоугольной или сферической

системе координат. Где располагается центр системы координат и каковы единицы измерения вдоль каждой оси, не очень важно. Важно то, что для отображения должны быть известны какие-то числовые значения координат отображаемых объектов.

```
#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CChildView

CChildView::CChildView()
{
}

CChildView::~CChildView()
{
}




// Обработчики сообщений CChildView

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)
{
if (!CWnd::PreCreateWindow(cs))
return FALSE;

cs.dwExStyle |= WS_EX_CLIENTEDGE;
cs.style &= ~WS_BORDER;
cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW|CS_VREDRAW|CS_DBLCLKS,
::LoadCursor(nullptr, IDC_ARROW),
reinterpret_cast<HBRUSH>(COLOR_WINDOW+1), nullptr);

return TRUE;
}

void CChildView::OnPaint()
{
CPaintDC dc(this); // контекст устройства для рисования

if (Index == 1) // режим отображения MM_TEXT
{
Graph.Draw(dc, 1, 1);
}

if (Index == 2)
{
```

```
Graph.GetRS(RS);
SetMyMode(dc, RS, RW);// Устанавливает режим отображения
MM_ANISOTROPIC (можно выбирать X и У )
Graph.Draw1(dc, 1, 1);
dc.SetMapMode(MM_TEXT);                    // Устанавливает режим
отображения MM_TEXT (напр X вправо, напр У вниз)
}
}

double CChildView::MyF1(double x)
{
double y = sin(x) / x;
return y;
}

double CChildView::MyF2(double x)
{
double y = pow(x, 3);
return y;
}

double CChildView::MyF3(double x)
{
double y = sqrt(x) * sin(x);
return y;
}

double CChildView::MyF4(double x)
{
double y = x * x;
return y;
}
```

3. Реализация третьего задания выполняется следующим образом

```
// Реализация карты сообщений
BEGIN_MESSAGE_MAP(CChildView, CWnd)
      ON_WM_PAINT()
      // сообщения меню выбора
      ON_COMMAND(ID_TESTS_F1, &CChildView::OnTestsF1)

      ON_COMMAND(ID_TESTS_F2, &CChildView::OnTestsF2)
      ON_COMMAND(ID_TESTS_F3, &CChildView::OnTestsF3)

      ON_COMMAND(ID_TESTS_F4, &CChildView::OnTestsF4)
      ON_COMMAND(ID_TESTS_F1234, &CChildView::OnTestsF1234)
```

```
END_MESSAGE_MAP()
```

Начинается карта сообщений с макроса **BEGIN_MESSAGE_MAP()**, у которого указывается два параметра: имя класса и имя класса родителя. Заканчивается карта сообщений макросом **END_MESSAGE_MAP()**.
CChildView имя класса, **CWnd-** имя класса-родителя. Директива **ON_COMMAND**, принимающая два параметра, где первый это айди функции и второй параметр функция, которая вызывается при обработке, отвечает за реализацию действий при выборе нужной кнопки**.**
ON_WM_PAINT() определяет функцию прорисовки экрана.

```
protected:
	afx_msg void OnPaint();
	DECLARE_MESSAGE_MAP()
public:
	// действия при выборе пункта меню
	afx_msg void OnTestsF1();
	afx_msg void OnTestsF2();
	afx_msg void OnTestsF3();
	afx_msg void OnTestsF4();
	afx_msg void OnTestsF1234();
```

В классе **CChildView** нужно определить следующие поля. Здесь 6 вируальных процедур на нажатие какой-либо из кнопок. Здесь, **afm_msg** является пустым макросом, но используется для того, чтобы показать нам, что функция является обработчиком сообщений.

4. Четвертое задание выполняется следующим образом:

```
void CChildView::OnTestsF1()	// MM_TEXT
{
	double Xl = -3 * pi;		// Координата X левого угла области
	double Xh = -Xl;		// Координата X правого угла области
	double dX = pi / 36;		// Шаг графика функции
	int N = (Xh - Xl) / dX;		// Количество точек графика
	X.RedimMatrix(N + 1);		// Создаем вектор с N+1 строками для
хранения координат точек по X
	Y.RedimMatrix(N + 1);		// Создаем вектор с N+1 строками для
хранения координат точек по Y
	for (int i = 0; i <= N; i++)
	{
		X(i) = Xl + i * dX;		// Заполняем массивы/векторы
значениями
		Y(i) = MyF1(X(i));
	}
```

```
	PenLine.Set(PS_SOLID, 1, RGB(255, 0, 0));  // Устанавливаем
параметры пера для линий (сплошная линия, толщина 1, цвет красный)
	PenAxis.Set(PS_SOLID, 2, RGB(0, 0, 255));  // Устанавливаем
параметры пера для осей (сплошная линия, толщина 2, цвет синий)
	RW.SetRect(100, 100, 500, 500);                  // Установка
параметров прямоугольной области для отображения графика в окне
	Graph.SetParams(X, Y, RW);                       // Передаем
векторы с координатами точек и область в окне
	Graph.SetPenLine(PenLine);                       // Установка
параметров пера для линии графика
	Graph.SetPenAxis(PenAxis);                       // Установка
параметров пера для линий осей
	Index = 1;                                             // Помечаем
для режима отображения MM_TEXT
	this->Invalidate();
}
```

Здесь вы можете увидеть переменную pi, которую мы инициализировали ранее(pi = 3.14159). Создаем интервал [-3pi; 3pi] простым присваиванием переменной **Xl = -3*pi** и **Xh = -Xl**(равносильно 3*pi). Таким же образом создаем переменную с шагом изменения аргумента.
Функция **RedimMatrix**, которая принимает один аргумент, изменяет размер матрицы с уничтожением данных. Передаваемый параметр – новое количество строк.
Для того чтобы заполнить наши векторы значениями нужно определить количество точек на заданном промежутке. Делается это с помощью формулы **(Xh-Xl)/dX**. Далее мы пишем цикл на количество итерация равным количеству точек на промежутке. X и Y наши векторы координат типа **CMatrix**. В этом классе мы перегружали различные операторы. Далее приведен кусок кода, который перегружает оператор «()» и позволяет получить нам i-ый элемент вектора:

```
double &CMatrix::operator()(int i)
// i - номер строки для вектора
{
	if (n_cols > 1)	//  Число столбцов больше одного
	{
		char* error = "CMatrix::operator(int): объект не вектор -
число столбцов больше 1 ";
		MessageBox(NULL, (LPCWSTR)error, L"Ошибка", MB_ICONSTOP);
		exit(1);
	}
	if (i > n_rows - 1)	//  проверка выхода за диапазон
	{
		TCHAR* error = TEXT("CMatrix::operator(int): выход индекса
за границу диапазона ");
		MessageBox(NULL, error, TEXT("Ошибка"), MB_ICONSTOP);
		exit(1);
```

```
	}
	return array[i][0];
}
```

Здесь происходит обработка ошибок, на случай, если несколько столбцов, а не один и проверка выхода за диапазон. Строка

return array[i][0]

возвращает нам i-ый элемент вектора, где array и представляет сам вектор.

Вернемся снова к нашей функции **OnTestsF1()**. Вектор X принимает значения от крайней точки нашего интервала и увеличивается каждый раз на шаг графика, умноженный i раз. А значение для Y устанавливается за счет написанной ранее функции **MyF1()**, принимающей в качестве параметра значения икса:

```
double CChildView::MyF1(double x)
{
	double y = sin(x) / x;
	return y;
}
```

Получив все значения и инициализировав наши вектора можно приступить к настройке параметров пера, которое было описано ранее. Таким образом, строка

PenLine.Set(PS_SOLID, 1, RGB(255, 0, 0))
говорит о том, что наше перо будет рисовать сплошной линией красного цвета толщиной 1 и **PenAxis.Set(PS_SOLID, 2, RGB(0, 0, 255))** – перо, которое рисует сплошной синей линией толщиной 2.

Следующая строка **RW.SetRect(100, 100, 500, 500)**, которая работает с объектом **RW** встроенного класс **CRect**. Метод **SetRect** позволяет установить нам прямоугольную область, передавая сначала в качестве параметров координаты верхней левой точки и далее нижней правой.

Graph – объект класса **CPlot2D**, который описан выше. Вначале с помощью метода **SetParams** мы передаем наши векторы X и Y и прямоугольную область, в которой и будет располагаться наш график. Далее благодаря методам **SetPenLine** и **SetPenAxis** мы устанавливаем параметры пера для графика функции и координатных осей. В конце устанавливаем переменной **Index** значение равное 1 для режима отображения MM_**TEXT** – каждый логический модуль преобразован в один пиксель устройства. Последней строкой мы запрещаем рисовать на нашей области что-то еще.

5-7. Выполнение пятого задания, как шестого и седьмого, и реализация функций **OnTestsF2()**, **OnTestsF3()**, **OnTestsF4(),** ничем не отличается от задания 4 и функции **OnTestsF1()**, кроме данных значений. Поэтому при выполнении заданий 5, 6, 7 руководствуйтесь заданием 4.
Небольшое дополнение: в 5 и 7 заданиях для переменной индекс используйте значение 2, таким образом вы будете использовать режим отображения **MM_ANISOTROPIC.**

8. Выполнение задания №8:

Выполнение восьмого задания заключается в объединении ваших прошлых успехов с заданиями 4, 5, 6, 7. Здесь поочередно рисуется каждый график и устанавливаются параметры для расположения его на экране.

```
void CChildView::OnTestsF1234()
{
	Invalidate();
	CPaintDC dc(this);
	double Xl = -3 * pi;
	double Xh = -Xl;
	double dX = pi / 36;
	int N = (Xh - Xl) / dX;
	X.RedimMatrix(N + 1);
	Y.RedimMatrix(N + 1);
	for (int i = 0; i <= N; i++)
	{
		X(i) = Xl + i * dX;
		Y(i) = MyF1(X(i));
	}
	PenLine.Set(PS_SOLID, 1, RGB(255, 0, 0));
	PenAxis.Set(PS_SOLID, 2, RGB(0, 0, 255));
	RW.SetRect(20, 10, 270, 260);
	Graph.SetParams(X, Y, RW);
	Graph.SetPenLine(PenLine);
	Graph.SetPenAxis(PenAxis);
Graph .Draw(dc, 1, 1);

	Xl = -5;
	Xh = -Xl;
	dX = 0.25;
	N = (Xh - Xl) / dX;
	X.RedimMatrix(N + 1);
	Y.RedimMatrix(N + 1);
	for (int i = 0; i <= N; i++)
	{
		X(i) = Xl + i * dX;
		Y(i) = MyF2(X(i));
```

```
        }
        PenLine.Set(PS_SOLID, 1, RGB(0, 255, 0));
        PenAxis.Set(PS_SOLID, 2, RGB(0, 0, 255));
        RW.SetRect(290, 10, 540, 260);
        Graph.SetParams(X, Y, RW);
        Graph.SetPenLine(PenLine);
        Graph.SetPenAxis(PenAxis);
        Graph.Draw(dc, 1, 1);

        Xl = 0;
        Xh = 6 * pi;
        dX = pi / 36;
        N = (Xh - Xl) / dX;
        X.RedimMatrix(N + 1);
        Y.RedimMatrix(N + 1);
        for (int i = 0; i <= N; i++)
        {
              X(i) = Xl + i * dX;
              Y(i) = MyF3(X(i));
        }
        PenLine.Set(PS_DASHDOT, 1, RGB(255, 0, 0));
        PenAxis.Set(PS_SOLID, 2, RGB(0, 0, 0));
        RW.SetRect(20, 280, 270, 530);
        Graph.SetParams(X, Y, RW);
        Graph.SetPenLine(PenLine);
        Graph.SetPenAxis(PenAxis);
        Graph.Draw(dc, 1, 1);

        Xl = -10;
        Xh = -Xl;
        dX = 0.25;
        N = (Xh - Xl) / dX;
        X.RedimMatrix(N + 1);
        Y.RedimMatrix(N + 1);
        for (int i = 0; i <= N; i++)
        {
              X(i) = Xl + i * dX;
              Y(i) = MyF4(X(i));
        }
        PenLine.Set(PS_SOLID, 2, RGB(255, 0, 0));
        PenAxis.Set(PS_SOLID, 2, RGB(0, 0, 255));
        RW.SetRect(290, 280, 540, 530);
        Graph.SetParams(X, Y, RW);
        Graph.SetPenLine(PenLine);
        Graph.SetPenAxis(PenAxis);
        Graph.Draw(dc, 1, 1);
}
```

Lab03.h

```
// Lab03.h: основной файл заголовка для приложения Lab03
//
#pragma once

#ifndef __AFXWIN_H__
	#error "включить stdafx.h до включения этого файла в PCH"
#endif

#include "resource.h"		// основные символы


// CLab01App:
// Сведения о реализации этого класса: Lab01.cpp
//

class CLab01App : public CWinApp
{
public:
	CLab01App() noexcept;

// Переопределение
public:
	virtual BOOL InitInstance();
	virtual int ExitInstance();

// Реализация

public:
	afx_msg void OnAppAbout();
	DECLARE_MESSAGE_MAP()
};

extern CLab01App theApp;
```

```
// Lab02.cpp: определяет поведение классов для приложения.
//

#include "stdafx.h"
#include "afxwinappex.h"
#include "afxdialogex.h"
#include "Lab03.h"
#include "MainFrm.h"


#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CLab01App

BEGIN_MESSAGE_MAP(CLab01App, CWinApp)
END_MESSAGE_MAP()

// Создание CLab01App

CLab01App::CLab01App() noexcept
{
```

```
		SetAppID(_T("Lab01.AppID.NoVersion"));
	}

	// Единственный объект CLab01App

	CLab01App theApp;


	// Инициализация CLab01App

	BOOL CLab01App::InitInstance()
	{
		// InitCommonControlsEx() требуются для Windows XP, если манифест
		// приложения использует ComCtl32.dll версии 6 или более поздней версии
для включения
		// стилей отображения.  В противном случае будет возникать сбой при
создании любого окна.
		INITCOMMONCONTROLSEX InitCtrls;
		InitCtrls.dwSize = sizeof(InitCtrls);
		// Выберите этот параметр для включения всех общих классов управления,
которые необходимо использовать
		// в вашем приложении.
		InitCtrls.dwICC = ICC_WIN95_CLASSES;
		InitCommonControlsEx(&InitCtrls);

		CWinApp::InitInstance();


		// Инициализация библиотек OLE
		if (!AfxOleInit())
		{
			AfxMessageBox(IDP_OLE_INIT_FAILED);
			return FALSE;
		}

		AfxEnableControlContainer();

		EnableTaskbarInteraction(FALSE);

		// Для использования элемента управления RichEdit требуется метод
AfxInitRichEdit2()
		// AfxInitRichEdit2();

		// Стандартная инициализация
		// Если эти возможности не используются и необходимо уменьшить размер
		// конечного исполняемого файла, необходимо удалить из следующего
		// конкретные процедуры инициализации, которые не требуются
		// Измените раздел реестра, в котором хранятся параметры
		// TODO: следует изменить эту строку на что-нибудь подходящее,
		// например на название организации
		SetRegistryKey(_T("Локальные приложения, созданные с помощью мастера
приложений"));


		// Чтобы создать главное окно, этот код создает новый объект окна
		// рамки, а затем задает его как объект основного окна приложения
		CFrameWnd* pFrame = new CMainFrame;
		if (!pFrame)
			return FALSE;
		m_pMainWnd = pFrame;
		// создайте и загрузите рамку с его ресурсами
		pFrame->LoadFrame(IDR_MAINFRAME,
			WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, nullptr,
			nullptr);
```

```
		// Разрешить использование расширенных символов в горячих клавишах меню
		CMFCToolBar::m_bExtCharTranslation = TRUE;



		// Одно и только одно окно было инициализировано, поэтому отобразите и
обновите его
		pFrame->ShowWindow(SW_SHOW);
		pFrame->UpdateWindow();
		return TRUE;
	}

	int CLab01App::ExitInstance()
	{
		//TODO: обработайте дополнительные ресурсы, которые могли быть добавлены
		AfxOleTerm(FALSE);

		return CWinApp::ExitInstance();
	}

	// Обработчики сообщений CLab01App


	// Диалоговое окно CAboutDlg используется для описания сведений о приложении

	class CAboutDlg : public CDialogEx
	{
	public:
		CAboutDlg() noexcept;

	// Данные диалогового окна
	#ifdef AFX_DESIGN_TIME
		enum { IDD = IDD_ABOUTBOX };
	#endif

	protected:
		virtual void DoDataExchange(CDataExchange* pDX);    // поддержка DDX/DDV

	// Реализация
	protected:
		DECLARE_MESSAGE_MAP()
	};

	CAboutDlg::CAboutDlg() noexcept : CDialogEx(IDD_ABOUTBOX)
	{
	}

	void CAboutDlg::DoDataExchange(CDataExchange* pDX)
	{
		CDialogEx::DoDataExchange(pDX);
	}

	BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
	END_MESSAGE_MAP()

	// Команда приложения для запуска диалога
	void CLab01App::OnAppAbout()
	{
		CAboutDlg aboutDlg;
		aboutDlg.DoModal();
	}
```

```
        // Обработчики сообщений CLab01App
```

Создадим графическую библиотеку LibChart2D:

LibChart2D:

```
#ifndef LIBGRAPH
#define LIBGRAPH 1
const double pi = 3.14159;
typedef double(*pfunc2)(double, double);    // Указатель на функцию

struct CSizeD
{
    double cx;
    double cy;
};
//---------------------------------------------------------------------------
struct CRectD
{
    double left;
    double top;
    double right;
    double bottom;
    CRectD() { left = top = right = bottom = 0; };
    CRectD(double l, double t, double r, double b);
    void SetRectD(double l, double t, double r, double b);
    CSizeD SizeD();
};

//---------------------------------------------------------------------------

CMatrix SpaceToWindow(CRectD& rs, CRect& rw);
// Возвращает матрицу пересчета координат из мировых в оконные
// rs - область в мировых координатах - double
// rw - область в оконных координатах - int
//---------------------------------------------------------------------------
void SetMyMode(CDC& dc, CRect& RS, CRect& RW);  //MFC
// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS -  область в мировых координатах - int
// RW -  Область в оконных координатах - int
//---------------------------------------------------------------------------
CMatrix CreateTranslate2D(double dx, double dy);
// Формирует матрицу для преобразования координат объекта при его смещении
// на dx по оси X и на dy по оси Y в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении начала
// системы координат на -dx оси X и на -dy по оси Y при фиксированном положении объекта
//---------------------------------------------------------------------------
CMatrix CreateTranslate3D(double dx, double dy, double dz);
// Формирует матрицу для преобразования координат объекта при его смещении
// на dx по оси X, на dy по оси Y,на dz по оси Z в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении начала
// системы координат на -dx оси X,на -dy по оси Y, на -dz по оси Z
// при фиксированном положени`и объекта
//---------------------------------------------------------------------------
CMatrix CreateRotate2D(double fi);
// Формирует матрицу для преобразования координат объекта при его повороте
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
```

```
// системы координат на угол -fi при фиксированном положении объекта
// fi - угол в градусах

//---------------------------------------------------------------------------
CMatrix CreateRotate3DZ(double fi);
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси Z
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси Z на угол -fi при фиксированном положении объекта
// fi - угол в градусах

//---------------------------------------------------------------------------
CMatrix CreateRotate3DX(double fi);
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси X
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси X на угол -fi при фиксированном положении объекта
// fi - угол в градусах

//---------------------------------------------------------------------------
CMatrix CreateRotate3DY(double fi);
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси Y
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси Y на угол -fi при фиксированном положении объекта
// fi - угол в градусах

//---------------------------------------------------------------------------
CMatrix CreateViewCoord(double r, double fi, double q);
// Создает матрицу пересчета точки из мировой системы координат в видовую
// (r,fi,q)- координата ТОЧКИ НАБЛЮДЕНИЯ(начало видовой системы координат)
// в мировой сферической системе координат( углы fi и q в градусах)
//---------------------------------------------------------------------------
CMatrix VectorMult(CMatrix& V1, CMatrix& V2);
// Вычисляет векторное произведение векторов V1 и V2
//---------------------------------------------------------------------------
double ScalarMult(CMatrix& V1, CMatrix& V2);
// Вычисляет скалярное произведение векторов V1 и V2
//---------------------------------------------------------------------------
double ModVec(CMatrix& V);
// Вычисляет модуль вектора V
//---------------------------------------------------------------------------
double CosV1V2(CMatrix& V1, CMatrix& V2);
// Вычисляет КОСИНУС угла между векторами V1 и V2
//---------------------------------------------------------------------------
double AngleV1V2(CMatrix& V1, CMatrix& V2);
// Вычисляет угол между векторами V1 и V2 в градусах
//---------------------------------------------------------------------------
CMatrix SphereToCart(CMatrix& PView);
// Преобразует сферические координаты PView  точки в декартовы
// PView(0) - r
// PView(1) - fi - азимут(отсчет от оси X), град.
// PView(2) - q - угол(отсчетот оси Z), град.
// Результат: R(0)- x, R(1)- y, R(2)- z
//---------------------------------------------------------------------------
void GetProjection(CRectD& RS, CMatrix& Data, CMatrix& PView, CRectD& PR);
// Вычисляет координаты проекции охватывающего фигуру паралелепипеда на
// плоскость XY в ВИДОВОЙ системе координат
// Data - матрица данных
// RS - область на плоскости XY, на которую опирается отображаемая поверхность
```

```
// PView - координаты точки наблюдения в мировой сферической системе координат
// PR - проекция
//---------------------------------------------------------------------------

#endif
```

LibGraph2D.cpp:

```
#include "stdafx.h"

CRectD::CRectD(double l, double t, double r, double b)
{
    left = l;
    top = t;
    right = r;
    bottom = b;
}
//---------------------------------------------------------------------------
void CRectD::SetRectD(double l, double t, double r, double b)
{
    left = l;
    top = t;
    right = r;
    bottom = b;
}

//---------------------------------------------------------------------------
CSizeD CRectD::SizeD()
{
    CSizeD cz;
    cz.cx = fabs(right - left); // Ширина прямоугольной области
    cz.cy = fabs(top - bottom); // Высота прямоугольной области
    return cz;
}

//---------------------------------------------------------------------------
CMatrix SpaceToWindow(CRectD& RS, CRect& RW)
// Возвращает матрицу пересчета координат из мировых в оконные
// RS - область в мировых координатах - double
// RW - область в оконных координатах - int
{
    CMatrix M(3, 3);
    CSize sz = RW.Size();     // Размер области в ОКНЕ
    int dwx = sz.cx;          // Ширина
    int dwy = sz.cy;          // Высота
    CSizeD szd = RS.SizeD(); // Размер области в МИРОВЫХ координатах

    double dsx = szd.cx;     // Ширина в мировых координатах
    double dsy = szd.cy;     // Высота в мировых координатах
    double kx = (double)dwx / dsx;    // Масштаб по x
    double ky = (double)dwy / dsy;    // Масштаб по y


    M(0, 0) = kx;  M(0, 1) = 0;     M(0, 2) = (double)RW.left - kx * RS.left;
    M(1, 0) = 0;   M(1, 1) = -ky;   M(1, 2) = (double)RW.bottom + ky * RS.bottom;
    M(2, 0) = 0;   M(2, 1) = 0;     M(2, 2) = 1;
    return M;
}
```

```
//---------------------------------------------------------------------------

void SetMyMode(CDC& dc, CRect& RS, CRect& RW)  //MFC
// Устанавливает режим отображения MM_ANISOTROPIC и его параметры
// dc - ссылка на класс CDC MFC
// RS -  область в мировых координатах - int
// RW -  Область в оконных координатах - int
{
    int dsx = RS.right - RS.left;
    int dsy = RS.top - RS.bottom;
    int xsL = RS.left;
    int ysL = RS.bottom;

    int dwx = RW.right - RW.left;
    int dwy = RW.bottom - RW.top;
    int xwL = RW.left;
    int ywH = RW.bottom;

    dc.SetMapMode(MM_ANISOTROPIC);
    dc.SetWindowExt(dsx, dsy);
    dc.SetViewportExt(dwx, -dwy);
    dc.SetWindowOrg(xsL, ysL);
    dc.SetViewportOrg(xwL, ywH);
}
//-----------------------------------------------------------------------------
CMatrix CreateTranslate2D(double dx, double dy)
// Формирует матрицу для преобразования координат объекта при его смещении
// на dx по оси X и на dy по оси Y в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении начала
// системы координат на -dx оси X и на -dy по оси Y при фиксированном положении объекта
{
    CMatrix TM(3, 3);
    TM(0, 0) = 1; TM(0, 2) = dx;
    TM(1, 1) = 1;  TM(1, 2) = dy;
    TM(2, 2) = 1;
    return TM;
}

//-----------------------------------------------------------------------------
CMatrix CreateTranslate3D(double dx, double dy, double dz)
// Формирует матрицу для преобразования координат объекта при его смещении
// на dx по оси X, на dy по оси Y,на dz по оси Z в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при смещении начала
// системы координат на -dx оси X,на -dy по оси Y, на -dz по оси Z
// при фиксированном положении объекта
{
    CMatrix TM(4, 4);
    for (int i = 0; i < 4; i++) TM(i, i) = 1;
    TM(0, 3) = dx;
    TM(1, 3) = dy;
    TM(2, 3) = dz;
    return TM;
}



//-----------------------------------------------------------------------------
CMatrix CreateRotate2D(double fi)
// Формирует матрицу для преобразования координат объекта при его повороте
```

```
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат на угол -fi при фиксированном положении объекта
// fi - угол в градусах
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0) * pi; // Перевод в радианы
    CMatrix RM(3, 3);
    RM(0, 0) = cos(ff); RM(0, 1) = -sin(ff);
    RM(1, 0) = sin(ff);  RM(1, 1) = cos(ff);
    RM(2, 2) = 1;
    return RM;
}
//---------------------------------------------------------------------------

CMatrix CreateRotate3DZ(double fi)
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси Z
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси Z на угол -fi при фиксированном положении объекта
// fi - угол в градусах
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0) * pi; // Перевод в радианы
    CMatrix RM(4, 4);
    RM(0, 0) = cos(ff); RM(0, 1) = -sin(ff);
    RM(1, 0) = sin(ff);  RM(1, 1) = cos(ff);
    RM(2, 2) = 1;
    RM(3, 3) = 1;
    return RM;
}
//---------------------------------------------------------------------------
CMatrix CreateRotate3DX(double fi)
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси X
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси X на угол -fi при фиксированном положении объекта
// fi - угол в градусах
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0) * pi; // Перевод в радианы
    CMatrix RM(4, 4);
    RM(0, 0) = 1;
    RM(1, 1) = cos(ff); RM(1, 2) = -sin(ff);
    RM(2, 1) = sin(ff); RM(2, 2) = cos(ff);
    RM(3, 3) = 1;
    return RM;
}
//---------------------------------------------------------------------------
CMatrix CreateRotate3DY(double fi)
// Формирует матрицу для преобразования координат объекта при его повороте вокруг оси Y
// на угол fi (при fi>0 против часовой стрелки)в фиксированной системе координат
// --- ИЛИ ---
// Формирует матрицу для преобразования координат объекта при повороте начала
// системы координат вокруг оси Y на угол -fi при фиксированном положении объекта
// fi - угол в градусах
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0) * pi; // Перевод в радианы
    CMatrix RM(4, 4);
```

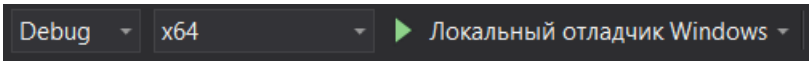
```
    RM(0, 0) = cos(ff); RM(0, 2) = sin(ff);
    RM(1, 1) = 1;
    RM(2, 0) = -sin(ff); RM(2, 2) = cos(ff);
    RM(3, 3) = 1;
    return RM;
}

//---------------------------------------------------------------------------
CMatrix VectorMult(CMatrix& V1, CMatrix& V2)
// Вычисляет векторное произведение векторов V1 и V2
{

    int b1 = (V1.cols() == 1) && (V1.rows() == 3);
    int b2 = (V2.cols() == 1) && (V2.rows() == 3);
    int b = b1 && b2;
    if (!b)
    {
        //char* error="VectorMult: неправильные размерности векторов! ";
        const TCHAR error[] = _T("VectorMult: неправильные размерности векторов! ");
        MessageBox(NULL, error, _T("Ошибка"), MB_ICONSTOP);
        exit(1);
    }
    CMatrix W(3);
    W(0) = V1(1) * V2(2) - V1(2) * V2(1);
    //double x=W(0);
    W(1) = -(V1(0) * V2(2) - V1(2) * V2(0));
    //double y=W(1);
    W(2) = V1(0) * V2(1) - V1(1) * V2(0);
    //double z=W(2);
    return W;
}

//---------------------------------------------------------------------------
double ScalarMult(CMatrix& V1, CMatrix& V2)
// Вычисляет скалярное произведение векторов V1 и V2
{
    int b1 = (V1.cols() == 1) && (V1.rows() == 3);
    int b2 = (V2.cols() == 1) && (V2.rows() == 3);
    int b = b1 && b2;
    if (!b)
    {
        char* error = "ScalarMult: неправильные размерности векторов! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double p = V1(0) * V2(0) + V1(1) * V2(1) + V1(2) * V2(2);
    return p;
}
//---------------------------------------------------------------------------
double ModVec(CMatrix& V)
// Вычисляет модуль вектора V
{
    int b = (V.cols() == 1) && (V.rows() == 3);
    if (!b)
    {
        char* error = "ModVec: неправильнfz размерность вектора! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double q = sqrt(V(0) * V(0) + V(1) * V(1) + V(2) * V(2));
    return q;
}
//---------------------------------------------------------------------------
```

```
double CosV1V2(CMatrix& V1, CMatrix& V2)
// Вычисляет КОСИНУС угла между векторами V1 и V2
{
    double modV1 = ModVec(V1);
    double modV2 = ModVec(V2);
    int b = (modV1 < 1e-7) || (modV2 < 1e-7);
    if (b)
    {
        char* error = "CosV1V2: модуль одного или обоих векторов < 1e-7!";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    int b1 = (V1.cols() == 1) && (V1.rows() == 3);
    int b2 = (V2.cols() == 1) && (V2.rows() == 3);
    b = b1 && b2;
    if (!b)
    {
        char* error = "CosV1V2: неправильные размерности векторов! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double cos_f = ScalarMult(V1, V2) / (modV1 * modV2);
    return cos_f;
}


//---------------------------------------------------------------------------
double AngleV1V2(CMatrix& V1, CMatrix& V2)
// Вычисляет угол между векторами V1 и V2 в градусах
{
    double modV1 = ModVec(V1);
    double modV2 = ModVec(V2);
    int b = (modV1 < 1e-7) || (modV2 < 1e-7);
    if (!b)
    {
        char* error = "AngleV1V2: модуль одного или обоих векторов < 1e-7!";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    int b1 = (V1.cols() == 1) && (V1.rows() == 3);
    int b2 = (V2.cols() == 1) && (V2.rows() == 3);
    b = b1 && b2;
    if (!b)
    {
        char* error = "AngleV1V2: неправильные размерности векторов! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double cos_f = ScalarMult(V1, V2) / (modV1 * modV2);
    if (fabs(cos_f) > 1)
    {
        char* error = "AngleV1V2: модуль cos(f)>1! ";
        //MessageBox(NULL, error, "Ошибка", MB_ICONSTOP);
        exit(1);
    }
    double f;
    if (cos_f > 0)f = acos(cos_f);
    else f = pi - acos(cos_f);
    double fg = (f / pi) * 180;
    return fg;
}
//---------------------------------------------------------------------------
CMatrix CreateViewCoord(double r, double fi, double q)
```

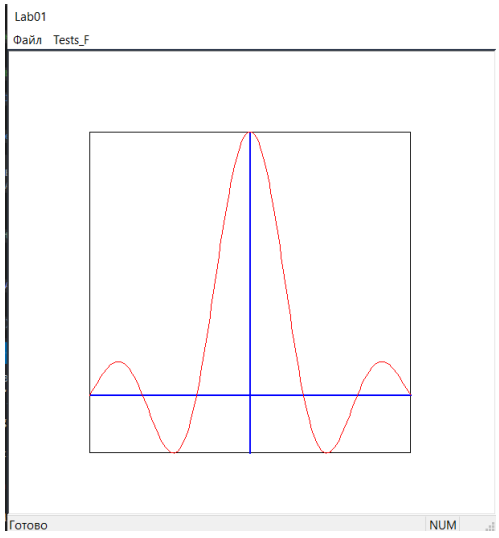
```
// Создает матрицу пересчета (4x4) точки из мировой системы координат в видовую
// (r,fi,q)- координата ТОЧКИ НАБЛЮДЕНИЯ(начало видовой системы координат)
// в мировой сферической системе координат( углы fi и q в градусах)
{
    double fg = fmod(fi, 360.0);
    double ff = (fg / 180.0) * pi; // Перевод в радианы
    fg = fmod(q, 360.0);
    double qq = (fg / 180.0) * pi; // Перевод в радианы

    CMatrix VM(4, 4);
    VM(0, 0) = -sin(ff);   VM(0, 1) = cos(ff);
    VM(1, 0) = -cos(qq) * cos(ff);  VM(1, 1) = -cos(qq) * sin(ff);  VM(1, 2) = sin(qq);
    VM(2, 0) = -sin(qq) * cos(ff);  VM(2, 1) = -sin(qq) * sin(ff);  VM(2, 2) = -cos(qq);
VM(2, 3) = r;
    VM(3, 3) = 1;
    return VM;
}

//---------------------------------------------------------------------------
CMatrix SphereToCart(CMatrix& PView)
// Преобразует сферические координаты PView  точки в декартовы
// PView(0) - r
// PView(1) - fi - азимут(отсчет от оси X), град.
// PView(2) - q - угол(отсчетот оси Z), град.
// Результат: R(0)- x, R(1)- y, R(2)- z
{
    CMatrix R(3);
    double r = PView(0);
    double fi = PView(1);                       // Градусы
    double q = PView(2);                        // Градусы
    double fi_rad = (fi / 180.0) * pi;  // Перевод fi в радианы
    double q_rad = (q / 180.0) * pi;        // Перевод q в радианы
    R(0) = r * sin(q_rad) * cos(fi_rad);    // x- координата точки наблюдения
    R(1) = r * sin(q_rad) * sin(fi_rad);    // y- координата точки наблюдения
    R(2) = r * cos(q_rad);                          // z- координата точки наблюдения
    return R;
}

//------------------------- GetProjection -----------------------------------

void GetProjection(CRectD& RS, CMatrix& Data, CMatrix& PView, CRectD& PR)
// Вычисляет координаты проекции охватывающего фигуру паралелепипеда на
// плоскость XY в ВИДОВОЙ системе координат
// Data - матрица данных
// RS - область на плоскости XY, на которую опирается отображаемая поверхность
// PView - координаты точки наблюдения в мировой сферической системе координат
// PR - проекция
{
    double Zmax = Data.MaxElement();
    double Zmin = Data.MinElement();
    CMatrix PS(4, 4);   // Точки в мировом пространстве
    PS(3, 0) = 1; PS(3, 1) = 1; PS(3, 2) = 1;   PS(3, 3) = 1;
    CMatrix MV = CreateViewCoord(PView(0), PView(1), PView(2)); //Матрица(4x4) пересчета

//в видовую систему координат
    PS(0, 0) = RS.left;
    PS(1, 0) = RS.top;
    PS(2, 0) = Zmax;

    PS(0, 1) = RS.left;
    PS(1, 1) = RS.bottom;
    PS(2, 1) = Zmax;
```

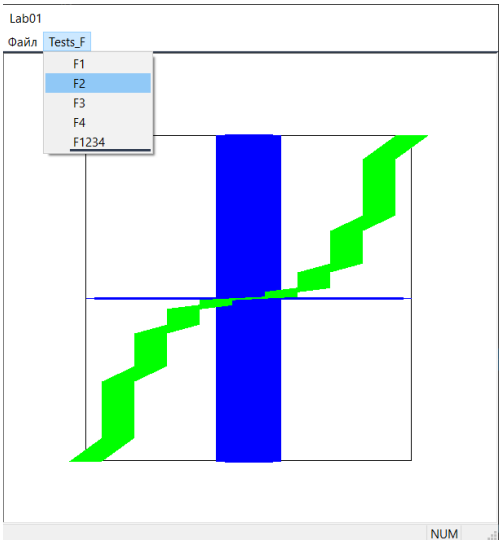
```
    PS(0, 2) = RS.right;
    PS(1, 2) = RS.top;
    PS(2, 2) = Zmax;

    PS(0, 3) = RS.right;
    PS(1, 3) = RS.bottom;
    PS(2, 3) = Zmax;

    CMatrix Q = MV * PS;       // Координаты верхней плоскости параллелепипеда в видовой
СК
    CMatrix V = Q.GetRow(0);         // Строка X - координат
    double Xmin = V.MinElement();
    double Xmax = V.MaxElement();
    V = Q.GetRow(1);            // Строка Y - координат
    double Ymax = V.MaxElement();

    PS(2, 0) = Zmin;
    PS(2, 1) = Zmin;
    PS(2, 2) = Zmin;
    PS(2, 3) = Zmin;

    Q = MV * PS;                   // Координаты нижней плоскости параллелепипеда в видовой
СК
    V = Q.GetRow(1);            // Строка Y - координат
    double Ymin = V.MinElement();
    PR.SetRectD(Xmin, Ymax, Xmax, Ymin);
}
```

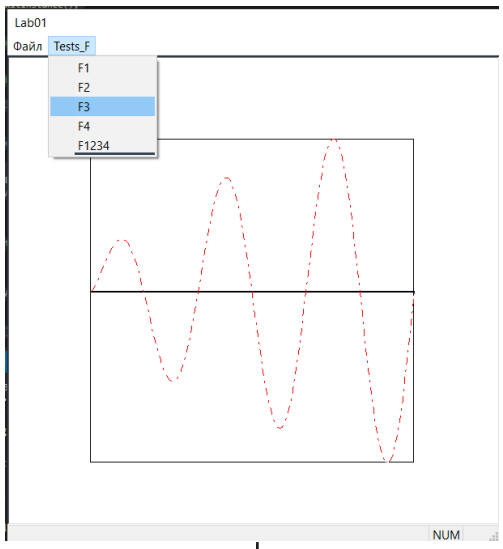
Теперь наше приложение готово к сборке.

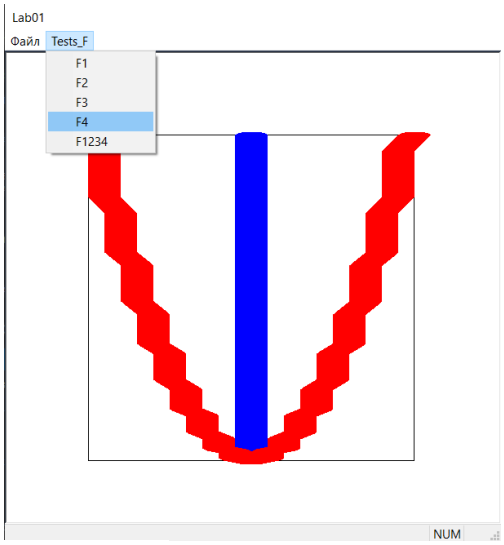
Запускаем отладчик: 

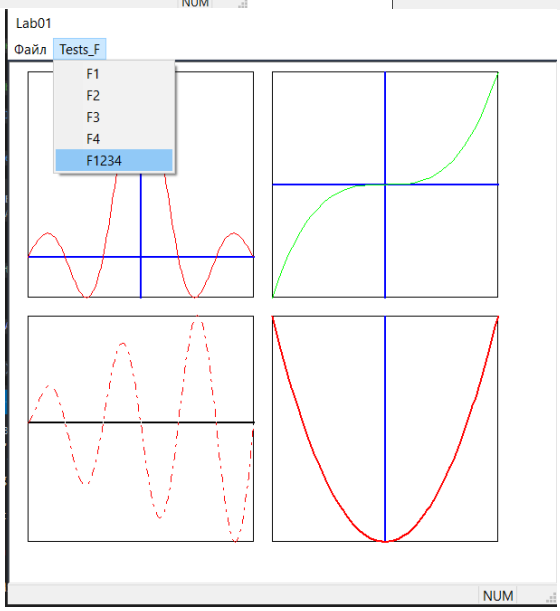
Получаем такой вот результат:











Литература:

Дятко, А.А., Мороз Л. С. Основы компьютерной геометрии и графики, учебно-методическое пособие для студентов учреждения высшего образования по направлению специальности 1-40 01 02-03 «Информационные системы и технологии (издательско-полиграфический комплекс»)», Белорусский государственный технологический университет, Минск : 2013. – 220 с.

Электронный учебно-методический комплекс(ЭУМК) «Компьютерная геометрия и графика»[Электронный ресурс]. Режим доступа: https:///www.belstu.by/faculties/fit/iikg/umk.html